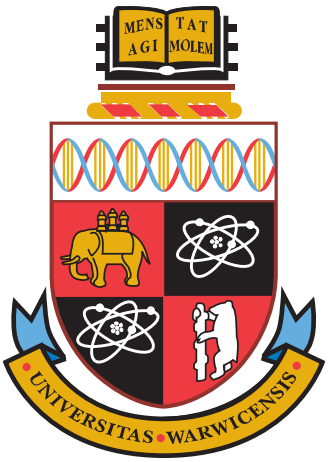


High-fidelity Graphics using Unconventional Distributed Rendering Approaches

Keith Bugeja
B.A. (Hons), M.IT.

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy in Engineering

School of Engineering
University of Warwick
2015

Contents

List of Figures

List of Tables

List of Publications

Papers

Bugeja, K., Debattista, K., Spina, S. & Chalmers, A. (2014). Collaborative high-fidelity rendering over peer-to-peer networks, *Eurographics Symposium on Parallel Graphics and Visualization*, The Eurographics Association, pp. 9-16.

Bugeja, K., Debattista, K., Spina, S. & Chalmers, A. (2014). High-fidelity graphics for dynamically generated environments using distributed computing, *6th International Conference on Games and Virtual Worlds for Serious Applications*.

Publications under preparation

Rendering as a Service (Chapter 5), for *Eurographics Parallel Graphics and Visualisation*

Remote asynchronous indirect lighting (Chapter 6), for *High Performance Graphics*

Peer-to-peer rendering (Chapter 8, extended), for *Transactions on Computer Graphics and Visualisation*

Acknowledgements

I would like to express my special appreciation and thanks to my advisors Kurt and Alan; without their guidance and persistent help this thesis would not have been possible. Kurt, you have been a tremendous mentor for me; thanks for encouraging my research and for allowing me to grow as a research scientist. Alan, your advice has been priceless. Thanks for your continuous support and enthusiasm.

Thanks also go to those in the Visualisation Group for providing such a varied and exciting research environment. Thanks go to Tom who provided some of the reference images used in this work, Jass who did the modelling on some of the scenes used, and all the others who are not listed here.

I would also like to thank my colleagues at the Department of Computer Science, particularly Sandro, whose insightful comments and suggestions made an enormous contribution to this work. I have also greatly benefitted from the stimulating discussions with my ex-colleagues Carmel, Gordon, Colin, Mark and François, to whom I am particularly grateful for their encouragement, suggestions and comments. Heartfelt thanks to Benjamin, Elvio and Lino for proofreading this dissertation.

Special thanks to my friends Kevin, Joe and Elvio, who are more like brothers than friends, and have been there for me through thick and thin.

My family has always supported me; words cannot express how grateful I am to my in-laws, Miriam and Frans, and my parents, Bernardina and Mario, for all the sacrifices they made on my behalf.

Finally, and most importantly, I would like to thank my beloved wife Marvic and our little ones, Sarah and Daniel, without whose great patience and support, this thesis would never have been possible.

Declaration

The work in this thesis is original and no portion of this work has been submitted in support of an application for another degree or qualification at this university or at another university or institution of learning.

Keith Bugeja

Abstract

High-fidelity rendering requires a substantial amount of computational resources to accurately simulate lighting in virtual environments. While desktop computing, with the aid of modern graphics hardware, has shown promise in delivering realistic rendering at interactive rates, real-time rendering of moderately complex scenes is still unachievable on the majority of desktop machines and the vast plethora of mobile computing devices that have recently become commonplace. This work provides a wide range of computing devices with high-fidelity rendering capabilities via oft-unused distributed computing paradigms. It speeds up the rendering process on formerly capable devices and provides full functionality to incapable devices. Novel scheduling and rendering algorithms have been designed to best take advantage of the characteristics of these systems and demonstrate the efficacy of such distributed methods. The first is a novel system that provides multiple clients with parallel resources for rendering a single task, and adapts in real-time to the number of concurrent requests. The second is a distributed algorithm for the remote asynchronous computation of the indirect diffuse component, which is merged with locally-computed direct lighting for a full global illumination solution. The third is a method for precomputing indirect lighting information for dynamically-generated multi-user environments by using the aggregated resources of the clients themselves. The fourth is a novel peer-to-peer system for improving the rendering performance in multi-user environments through the sharing of computation results, propagated via a mechanism based on epidemiology. The results demonstrate that the boundaries of the distributed computing typically used for computer graphics can be significantly and successfully expanded by adapting alternative distributed methods.

Keywords: computer graphics, distributed computing, interactive global illumination

CHAPTER 1

Introduction

In the field of computer graphics, rendering is the process by which a virtual scene containing mathematical representations of an environment is synthesised into an image. These representations typically include geometry, material properties of surfaces, lighting conditions and camera attributes. High-fidelity rendering concerns itself with the generation of images that are virtually indistinguishable from real world images (Greenberg *et al.*, 1997); see Figure 1.1 for examples. Thus, the physical behaviour of light is simulated as accurately as possible in an effort to model real world phenomena and replicate them in the final synthesised image. The essence of high-fidelity rendering is captured by the rendering equation (Kajiya, 1986), an integral equation that can be used to evaluate lighting at a specific point in an environment. The rendering equation cannot be solved analytically because of the complexity of the models involved. Numerical simulations based on the finite element or Monte Carlo methods are employed instead. High-fidelity rendering is a computationally expensive process and, as is often the case with such physical simulations, parallel and distributed computing are introduced to speed up computation time or solve larger problem sets. This thesis investigates the challenges of using a number of distributed computing approaches that are not conventionally employed in computer graphics, with two primary aims: to democratise high-fidelity rendering over a large swath of computing devices that mostly lack the power for such visualisation; and to speed up traditional rendering algorithms via these unconventional approaches.

Figure 1.1: Examples of high-fidelity rendering.

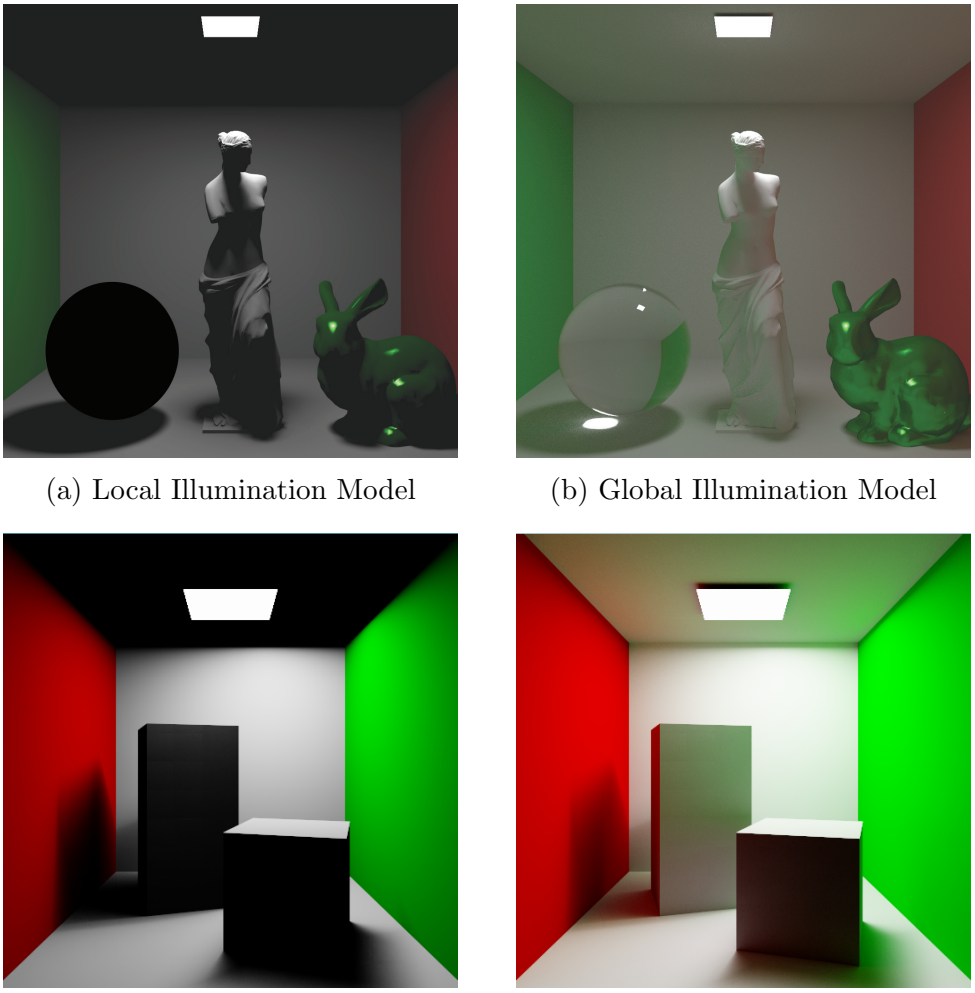
1.1 High-fidelity Rendering

Following the real-life process by which an image sensor converts an optical image into an electronic signal, the simulation of light transport in high-fidelity rendering calculates lighting from the point of view of a virtual camera. The interaction of light with the environment is typically classified into two types of contributions, direct and indirect (§2.5.3). The direct contribution models light directly incident on surfaces, that is, light emitted from a source that interacts at most once with a surface before reaching the virtual camera. The indirect contribution captures light that has interacted with multiple surfaces prior to reaching the virtual camera. Lighting models that are based on the direct contribution alone are referred to as local illumination models, while models that aggregate direct and indirect lighting are called global illumination (GI) methods. Local illumination models can be computed quickly and efficiently, and have been predominantly used in real-time rendering. On the other hand, GI is more complex due to the myriad of possible interactions of light between different surfaces in a scene. In high-fidelity image synthesis, the modelling of GI is essential for physical correctness and to bolster realism.

Figure 1.2 shows the Cornell box, a common data set used in computer graphics, synthesised using direct lighting (figures 1.2a, 1.2c) and GI (figures 1.2b, 1.2d). The images rendered using direct lighting alone fail to account for a number of phenomena, such as caustics and colour bleeding. A caustic is the projection of an envelope of light rays reflected or refracted by a specular curved object onto another surface. An example can be seen in the bright spot under the sphere of Figure 1.2b. Colour bleeding is caused by light interreflections between surfaces, where colour is transferred between nearby objects. Figure 1.2d illustrates colour bleeding where the green and red walls reflect coloured light onto the boxes, floor and ceiling.

1.1.1 Applications

The realism afforded by high-fidelity rendering and GI algorithms is highly sought-after by fields that require not just plausible rendering but physically-correct simulations of light transport. Although traditionally constrained by the time required to synthesise an image, advances in both rendering techniques and computer hardware have thus made the scope of high-fidelity rendering considerably



(a) Local Illumination Model (b) Global Illumination Model

(c) Local Illumination Model (d) Global Illumination Model

Figure 1.2: Examples of different illumination models.

broader in application (Dutre *et al.*, 2003).

Architecture

In the design of buildings, an accurate simulation of illumination is vital to predict how a structure will look under specific lighting conditions. This includes both indoor illumination, where physically-based rendering becomes an important tool for engineers in the design of lighting, and outdoor, which allows visualisation of buildings under different atmospheric conditions at any time of the day or year. High-fidelity rendering as a tool allows an architect to preview and alter building designs, permitting validation prior to construction. In the case of up-front investments, interactive walkthroughs can give a very good idea to potential clients of what they would be buying

into (Yan *et al.*, 2011).

Cultural Heritage

In cultural heritage studies, sites and artefacts are digitised for the purposes of preservation and study, some of which are also reconstructed and relighted with authentic lighting conditions to provide greater accuracy in the recreation of the past (Happa *et al.*, 2010; 2012).

Computer Games

Modern game engines have recently started integrating techniques to approximate dynamic indirect lighting in real-time, improving the realism and immersion provided by game experiences (Myszkowski *et al.*, 2001). The use of pre-rendered cinematic sequences, very often using physically-based rendering techniques, is also very common. The strive for realism in video games has set a trend that will continue for the foreseeable future in which a more complete global illumination solution is sought.

Film

In computer generated films or visual effects in general, global illumination is used to provide a consistent lighting representation. Photorealistic animated characters, which have come to replace animatronics in many productions, often need to be overlaid onto existing film, and actors are frequently cast on green screen and superimposed over computer generated imagery; these benefit greatly from the use of physically-based rendering to maintain an accurate and consistent portrayal of lighting across both captured and the synthesised imagery. Non-photorealistic rendering also benefits from the use of physically-based methods in the creation of plausible and aesthetically pleasing lighting representations (Tabellion & Lamorlette, 2004).

Simulation and Training

Realistic rendering and accurate lighting is also very important for training simulators. For instance, flight and driving simulators benefit from a realistic portrayal of the environment that corresponds as much as possible to the simulated real world scenario. Training a pilot for adverse weather conditions requires accurate atmospheric visualisation, while training for night driving requires capturing aspects of lighting such as street lights, lighting from other vehicles and so on.

Design

Predictive simulations of cars, appliances, consumer electronics and so on simulate how an object will look in a real or virtual environment; this is extremely useful in that these objects can be assessed aesthetically without the need to build possibly expensive prototypes.

Healthcare

Simulated surgical training and virtual reality-based medical simulations can provide a safe learning environment for repeated practice of procedures (Kneebone, 2003). The use of high-fidelity graphics in these scenarios can narrow the visual gap between the simulated and actual environment, increasing the realism, and possibly the effectiveness of the training. Realistic training simulations can potentially cut the increasing costs of surgical training in medical education, especially in the light of the constant development of new surgical procedures (Marks *et al.*, 2007).

Whether to further hypothesis evaluation in archaeology, provide training in a safe and controlled environment, reduce the cost of prototyping through predictive simulation, or enhance immersion via realism in video games, high-fidelity rendering is a crucial common denominator and any advancement that furthers the field would be beneficial to its areas of application.

1.2 Distributed Rendering Approaches

The computational complexity of high-fidelity rendering precludes its use on anything but powerful desktop machines and specialised supercomputer clusters. In a real-time setting, approximations that trade accuracy for speed allow for GI to be computed at interactive rates. Usually, these trade-offs come in terms of a limited number of bounces of indirect lighting, or constraints on scene geometry, materials or other lighting properties. Even then, dynamic GI requires a powerful desktop machine with a good graphics processing unit (GPU) in order to simulate moderately complex environments.

1.2.1 Research Methodology

Andrews (1991) enumerates five principal paradigms in the concurrent programming landscape, with *client-server* and *collaborating peers* being identified as the

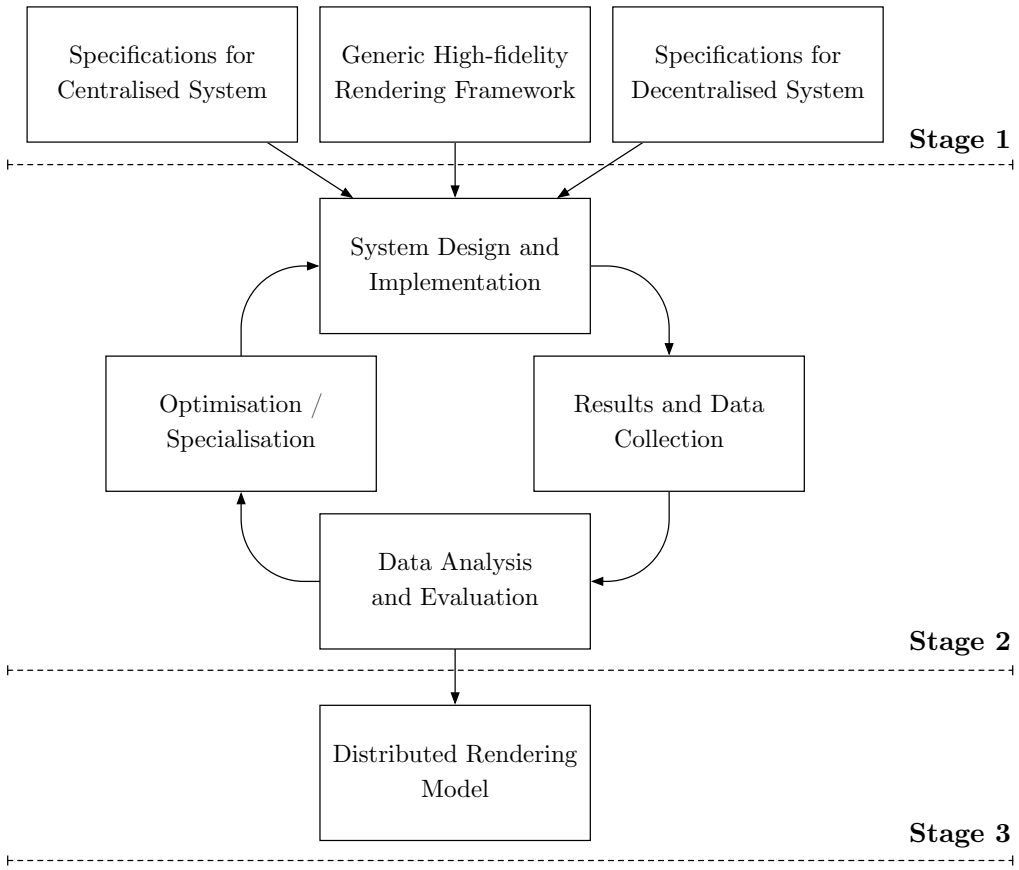


Figure 1.3: Research Methodology

main distributed computing models. In the client-server model, the roles of both client and server are clearly delineated, with the server providing a resource and clients consuming it. This resource can be a service or function provided in response to a request initiated by a client. The model is inherently centralised with computing carried out at a central location, allowing work to be offloaded from clients in a shift commensurate with the processing power of clients and server. Thus, clients can be weak and underpowered, with the server compensating through the use of powerful hardware.

The collaborating peers model, henceforth referred to as *peer-to-peer*, is antithetical to client-server in terms of resource centralisation, supply and consumption. Peers are considered equal and are equally powerful participants in the distributed program or application. In contrast to client-server, resources are not supplied through a centralised system but are rather pooled amongst peers, which not only act as consumers but also as providers.

Both client-server and peer-to-peer models are pillars of distributed comput-

ing and have been assumed as the starting point in the design of the algorithms put forward by this work. These two models characterise the extremes of the centralisation/decentralisation spectrum and their choice is motivated by the possibility of a methodical exploration of the continuum they comprise. Novel algorithms could be expressed in terms of various degrees of hybridisation of these two extremes. The exploration of this algorithmic space has been guided by a number of properties of distributed computing and rendering, which are introduced and discussed at length in §4.7. These properties serve to demonstrate the gap in knowledge and state of the art.

An overview of the approach taken in the research, design, implementation and evaluation of the methods presented in this work is illustrated in Figure 1.3. The adopted methodology draws heavily from parallels in software development where requirements and solutions evolve through adaptive planning and continuous improvement. The problem of distributed rendering is two-faceted: on one hand is the solution of the rendering equation, on the other the efficient decomposition of the rendering problem in terms of distributed programs and applications. A substantial body of research exists on techniques and methods for providing efficient solutions to the rendering equation (see Chapter 3), but in terms of distributed rendering, only a sliver of the continuum mentioned above has been explored (see Chapter 4). Initially, a generic high-fidelity rendering framework was implemented, based on the state of the art. Specifications for client-server and peer-to-peer distributed systems were devised, focussing on the management of server resources in the first instance, and the ordering of information dissemination for a consistent shared state in the second. These three tasks, shown in the first stage of Figure 1.3, seeded the second stage of research and development, where the specifications in conjunction with the rendering framework were used to draw system designs and the respective implementations. The second stage is iterative in nature; each implementation is evaluated, with the resulting feedback employed in their refinement and optimisation. The insight gained from each iteration has been used to address limitations in either method or implementation via optimisation, and in some cases, to develop specialisations of the methods, tailored for specific contexts. Optimisations improve the overall system operation and result in a more efficient execution, while specialisations entail a reconfiguration of the distributed rendering pipeline such that it becomes either more centralised or more decentralised. For the purpose of evaluation, data set selection generally mirrors previous work on high-fidelity rendering, such as

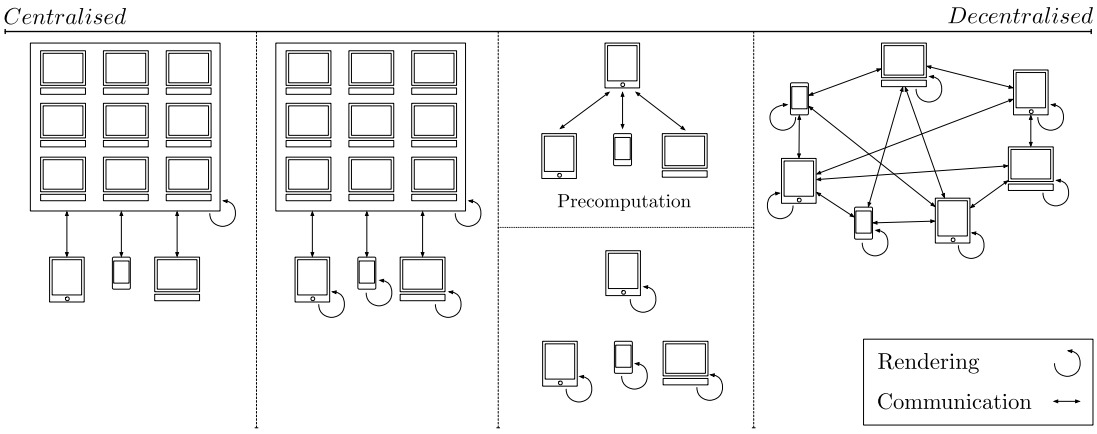
Aggarwal *et al.* (2012), Debattista *et al.* (2011) and Wald, Kollig, Benthin, Keller & Slusallek (2002). The evaluation criteria are defined in terms of the properties introduced in §4.7; not all methods satisfy all properties. General properties such as scalability and speed-up feature across all evaluations.

This approach led to the four distributed rendering techniques presented in this work (§1.3). There is chronological overlap in the second stage of the research methodology between different techniques; initially, the centralised and decentralised methods were being developed concurrently. In this dissertation, the presentation forfeits the chronological order of development in favour of a logical progression from centralised to decentralised techniques.

1.3 Research Outlook

Primarily, the work looks into two unexplored aspects of distributed computation for computer graphics, firstly democratising high-fidelity rendering via a service model; and concurrently the decentralisation attributes of distributed computing are also explored. In order to show the efficacy of unconventional distributed approaches, four methods that employ distinct characteristics of these paradigms will be presented.

Client-server and service-oriented architectures provide a means to build applications from existing software services, which are seen as black boxes. In order to make interactive high-fidelity rendering available to a wide spectrum of devices, it was elected to abstract the rendering quality from the capabilities of the target device and provide rendering as a service (RaaS), which is streamed to clients over the network. The focus of RaaS is that of providing an efficient use of server resources across multiple connected clients. Remote rendering places undue dependence on the network infrastructure connecting clients and server, which is a problem in highly-interactive applications. High-definition (HD) and higher resolutions exacerbate this problem by putting further strain on the network due to increased bandwidth requirements. These problems are addressed via Remote Asynchronous Indirect Lighting (RAIL). RAIL extends RaaS by decoupling more expensive lighting computations from the rest of the rendering, which is moved to the client device. The server asynchronously streams lighting information to the client in a format that is independent of screen resolution, allowing the client to render at HD resolutions without necessarily increasing



(a) RaaS (b) RAIL (c) PPIL (d) PePeR

Figure 1.4: Spectrum of distributed computing architectures. The four methods proposed by this work are shown spanning the spectrum from entirely centralised (left) to totally decentralised (right).

bandwidth requirements. In both these approaches, powerful remote resources are assumed which bear the brunt of the computations.

Decentralisation waives the dependency on powerful central systems by breaking down their functionality and redistributing it across peers. In a step towards decentralisation, the algorithms employed by RAIL were adapted and applied to the aggregated resources of client devices, to precompute per-vertex indirect lighting (PPIL) for a multi-user environment. Finally, via peer-to-peer rendering (PePeR), the core concept of PPIL is extended to dynamic environments and networks by transitioning towards a fully decentralised approach to computation and making away with any precomputation. Furthermore, PePeR is cognate with RaaS in that it can support multiple rendering techniques.

1.3.1 Rendering as a Service (RaaS)

The first approach in this thesis focuses on a service-based model for interactive high-fidelity rendering. Inspired by the *Software as a Service* (SaaS) model (Mell & Grance, 2011), this approach delivers interactive high-fidelity graphics to multiple concurrent clients whose computational power is immaterial to the fidelity of the rendering. This mirrors the ability of a client device, such as a smartphone or tablet, to consume a cloud service and perform an operation that due to resource constraints would otherwise be impossible to carry out locally.

The concept of offloading rendering to the cloud has emerged with the es-

tablishment of cloud render farms. These data centres are governed by a batch system that allows rendering jobs to be submitted and queued, and only scheduled when enough resources are available. Tools for monitoring rendering jobs are usually provided, but the whole process simply extends traditional non-interactive rendering by offloading computation to powerful clusters in the cloud and adding metering for usage and costings. Indeed, such systems are scalable and elastic, but the lack of interactivity does nothing to elicit response times to system changes in the sub-second region; thus the lack of interactivity is an issue. Interactive control of remote machines has long been a possibility with remote access protocols that allow output from graphical user interfaces to be streamed over the network (and the Internet) to a user on a controlling machine. In the last decade, this idea has been successfully extended to support graphically rich applications such as interactive rendering and video games, albeit contrary to cloud render farms, such systems do not consider horizontal scaling and elasticity to be as important. They focus on efficient transport and streaming of images over various networks of different capabilities, instead.

In this work, an attempt is made to fill the gap in the status quo by proposing *Rendering as a Service* (RaaS), a framework for interactive high-fidelity rendering in the cloud, with the aim of striking a balance between scalability, elasticity and interactivity.

Objectives

- to establish RaaS, a framework for interactive high-fidelity rendering
- to evaluate the scalability and elasticity aspects of RaaS

1.3.2 Rendering Asynchronous Indirect Lighting (RAIL)

The second approach presented looks at the problem of network performance in interactive remote rendering solutions. The network connecting clients and service provider plays an important part in the performance of such systems. Fluctuations in network service affect both responsiveness and the quality of the presented graphics. These problems are exacerbated when further stress is placed on the network infrastructure to stream at resolutions higher than the 720p employed by most of these services. This is a serious consideration in the light of emerging 4K (3840×2160) and 5K (5120×2880) display resolutions and

applications that take advantage of the higher definition.

To counter the problem of network latency and bandwidth impacting on the responsiveness of high-fidelity rendering applications, a novel algorithm for *Rendering Asynchronous Indirect Lighting* (RAIL) is proposed. RAIL aims to address problems with responsiveness by decoupling stages of the rendering pipeline, where fast computations are carried out on the local device, while expensive computations are carried out in the cloud. The two end points are kept consistent using an asynchronous messaging model. RAIL also addresses the problem of increased bandwidth when streaming at higher resolutions by introducing a lightweight object-space, resolution-independent data structure for the synchronisation of client and server. The algorithm used in the reconstruction of indirect lighting on the client devices is also lightweight and scalable, and can be deployed across a diverse spectrum of devices, from tablets to desktop machines.

Objectives

- to extend RaaS with RAIL, for low-latency, low-bandwidth fully interactive rendering
- to evaluate RAIL for image quality, server scalability and network performance

1.3.3 Precomputed Per-vertex Indirect Lighting (PPIL)

The third method presented scales the global illumination algorithm introduced in RAIL to augment dynamically-generated multi-user environments with precomputed indirect lighting information. Systems that lack the computational power required to render a dynamic global illumination solution may resort to static lighting solutions. Indirect lighting information is precomputed in an offline process and merged at runtime with dynamic direct lighting. Devices without significant computational power cannot precompute indirect lighting for fully dynamically-generated environments. Although this may hold for each device individually, the aggregate computational power of all clients may instead prove sufficient.

An algorithm for *Precomputed Per-vertex Indirect Lighting* (PPIL) is proposed; the method takes advantage of distributed computing to exploit the computational resources of individual participants in a multi-user environment to

accelerate the lighting precomputation process for dynamically-generated environments.

Objectives

- to make available the global illumination algorithm in RAIL for devices with limited GPU support
- to exploit distributed computing in the precomputation of indirect lighting for dynamically-generated scenes
- to evaluate the performance and quality of the global illumination algorithm against a straightforward approach

1.3.4 Peer-to-peer Rendering (PePeR)

The fourth approach presented in this thesis looks at the application of high-fidelity rendering to fully decentralised networking environments. Peer-to-peer (P2P) computing has arisen as one of the major models for offloading computation from a centralised server, to benefit a number of peers participating in a common activity. It has a strong association with content delivery and in the context of multimedia, it finds frequent use in the streaming of video and audio. The use of P2P models carries a number of advantages in terms of scalability, burden-sharing and fault-tolerance, while eliminating the troublesome single point of failure in client-server systems. Nevertheless, moving to this amorphous environment introduces a number of challenges, such as efficient searching for required content, ensuring propagation and consistency of created content, the problem of churn, where peers frequently join and leave the network in large numbers, and so on.

A novel framework for *Peer-to-peer Rendering* (PePeR) is introduced, which addresses the issues raised above. PePeR is first and foremost a specification for sharing computation over unstructured P2P networks. It provides an abstract event system that can discriminate between private (*internal*) and public (*observable*) events that are totally ordered and used to access and modify shared data structures. PePeR uses epidemiological models to disseminate information and propagate events to network peers. The problem of churn is addressed via an anti-entropy peer exchange system. The popular irradiance cache algorithm

by Ward *et al.* (1988) is used with PePeR to provide collaborative high-fidelity rendering in a multi-user virtual environment.

Objectives

- to provide a specification for collaborative high-fidelity rendering over peer-to-peer networks
- to evaluate the computation gains from peer-to-peer collaboration

1.4 Thesis Outline

This thesis is organised as follows:

Chapter 2: Background provides an overview of the concepts related to high-fidelity rendering such as radiometry, the various formulations of the rendering equation, and Monte Carlo methods, amongst others.

Chapter 3: High-fidelity Rendering provides a detailed literature review of the various algorithms used in the computation of global illumination, both on CPU and GPU, for point-based and finite element methods.

Chapter 4: Parallel and Distributed Rendering extends the literature review with a discussion of parallel and distributed rendering methods on hardware ranging from dedicated supercomputer clusters to cloud infrastructures. Both offline and interactive rendering techniques are discussed.

Chapter 5: Rendering as a Service (RaaS) introduces the concept of interactive physically-based rendering as a service, proposing a scalable and elastic method for remote rendering.

Chapter 6: Remote Asynchronous Indirect Lighting (RAIL) proposes a novel method for the asynchronous remote computation of indirect lighting in multi-user environments.

Chapter 7: Precomputed Per-vertex Indirect Lighting (PPIL) introduces a method for precomputing indirect lighting for dynamically-generated multi-user environments using the aggregated resources of client devices.

Chapter 8: Peer-to-peer Rendering (PePeR) presents a novel method for sharing computation such as indirect lighting contribution via collaboration in a decentralised and unstructured peer-to-peer network.

Chapter 9: Conclusions

This chapter concludes the dissertation, discussing limitations of this work and presenting potential avenues for future work.

CHAPTER 2

Background

This chapter introduces a number of tools that are essential in gaining an understanding of the process of physically-based rendering. The chapter opens with a brief overview of radiometry, together with surface reflection and transmission functions, which are introduced with a slant towards their use in computer graphics. Subsequently, the rendering equation is introduced and presented in three of its most common formulations. A short overview of Monte Carlo methods is also provided, given that this work mostly deals with solving the rendering equation using such methods. Finally, a common representation of spectral power distributions used in computer graphics is provided, and operators for the manipulation of collections of spectra given.

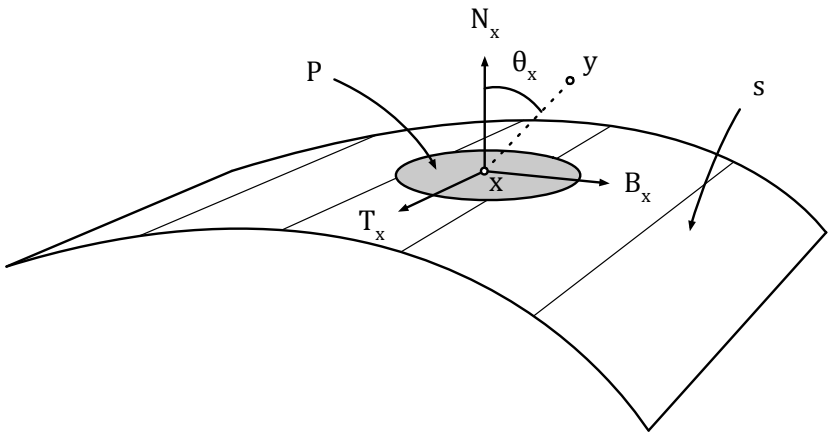


Figure 2.1: Geometric properties of a point $\mathbf{x}$ on a surface.

2.1 Preliminaries

A scene $\mathbb{S}$ is a collection of surfaces where $\mathbf{x} : S(\mathbf{x}), \mathbf{x} \in \mathbb{R}^3$ is a point on a surface $s \in \mathbb{S}$. For each point $\mathbf{x} : S(\mathbf{x})$, an orthonormal basis O_x is defined such that $O_x = (T_x, B_x, N_x)^T$, where N_x is the surface normal, B_x is the binormal and T_x the tangent at $\mathbf{x}$, and $N_x, B_x, T_x \in \mathbb{R}^3$. The vector pair (B_x, T_x) forms a tangent plane P to N_x, and for any point $\mathbf{y}$ on P, $(\mathbf{y} - \mathbf{x}) \cdot N_x = 0$, where $\cdot$ is the scalar product. This is generalised to specify the relationship of any point $\mathbf{y} \in \mathbb{R}^3$ to P; let $v = \frac{\mathbf{y}-\mathbf{x}}{|\mathbf{y}-\mathbf{x}|}$ be the normalised direction vector from $\mathbf{x}$ to $\mathbf{y}$: $\mathbf{y}$ is above P if $(v \cdot N_x) > 0$ and below if $(v \cdot N_x) < 0$. Moreover, $v \cdot N_x = \cos\theta_x$, where θ_x is the angle v makes to N_x (see Figure 2.1). All direction vectors are assumed to be normalised (i.e., of unit length).

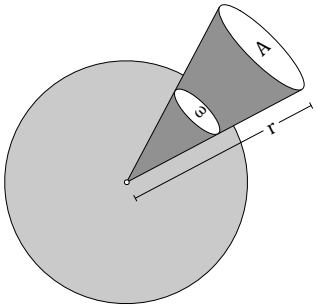


Figure 2.2: Representation of a solid angle.

2.2 Solid Angles

A point on a surface is enclosed in a sphere of directions along which light can arrive or leave. The pattern a source of light generates on the sphere dictates the effect it has on the surface point. A useful measure on the surface of a unit sphere is the solid angle, or the surface area subtended by a two-dimensional angle (Figure 2.2). It is defined as:

$$\omega = \frac{A}{r^2}. \tag{2.1}$$

The solid angle of an object is equal to the area of the segment of a unit sphere centred at the angle's vertex that the object covers. A differential solid angle with apex at $\mathbf{x}$ for a differential surface patch $\mathrm{d}A_y$ at some point $\mathbf{y}$ (Figure

2.3), is defined as:

$$\mathrm{d}\omega = \frac{\mathrm{d}A_y \ \cos\theta}{r^2} \tag{2.2}$$

$$= \frac{\mathrm{d}A_y \ (N_y \cdot \omega_y)}{r^2}, \tag{2.3}$$

where θ is the angle between the normal N_y of the differential surface $\mathrm{d}A_y$ and the direction ω_y, from $\mathbf{x}$ to $\mathbf{y}$, the centroid of $\mathrm{d}A_y$, and $r = |\mathbf{y} - \mathbf{x}|$. The area $\mathrm{d}A \cdot \cos\theta$ is the intercepted sphere surface; by definition, the solid angle is either a cone or a pyramid.

A direction vector ω over the unit hemisphere can be represented in a spherical coordinate system using a pair of angles (θ, ϕ), where θ is the polar angle ($0 \leq \theta \leq \pi$) and ϕ the azimuthal angle ($0 \leq \phi \leq 2\pi$) (Figure 2.4a). The pair (θ, ϕ) may be converted into a Cartesian coordinate triple (x, y, z), where $x^2 + y^2 + z^2 = 1$, using:

$$x = \cos\phi \ \sin\theta \tag{2.4}$$

$$y = \sin\phi \ \sin\theta \tag{2.5}$$

$$z = \cos\theta \tag{2.6}$$

A differential solid angle can also be expressed in terms of the angles θ and ϕ in a spherical coordinate system (Figure 2.4b):

$$\mathrm{d}\omega = \sin\theta \ \mathrm{d}\theta \ \mathrm{d}\phi. \tag{2.7}$$

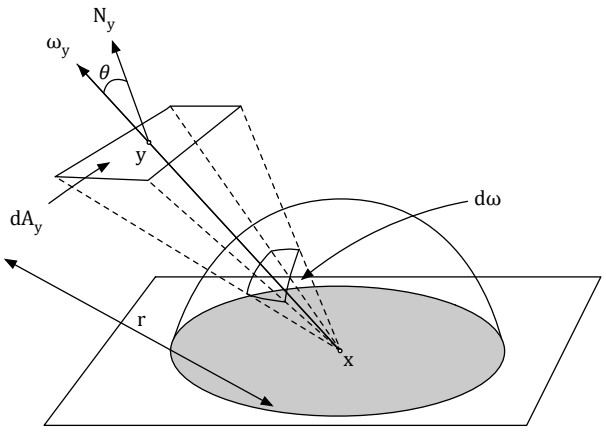
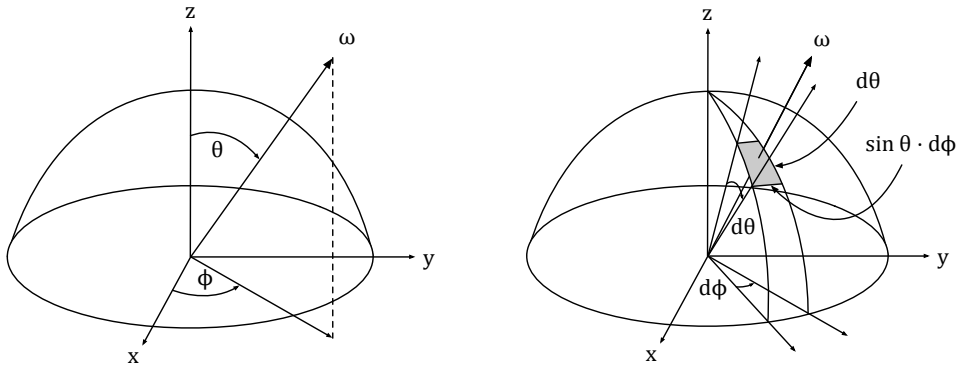


Figure 2.3: Representation of a differential solid angle.



(a) Spherical coordinate representation (b) Solid angle in spherical coordinates

Figure 2.4: Differential solid angle representation in spherical coordinates.

A solid angle Ω subtending an entire sphere is given by:

$$\Omega = \int_0^{2\pi} \int_0^{\pi} \sin\theta \ \mathrm{d}\theta \ \mathrm{d}\phi = 4\pi, \tag{2.8}$$

while a solid angle Ω_H subtending an entire hemisphere is given by:

$$\Omega_H = \int_0^{2\pi} \int_0^{\frac{\pi}{2}} \sin\theta \ \mathrm{d}\theta \ \mathrm{d}\phi = 2\pi. \tag{2.9}$$

The unit of the solid angle is the steradian [sr] and is dimensionless. In this work, Ω and Ω_H are used interchangeably to define a solid angle subtending an entire hemisphere, unless explicitly stated.

2.3 Radiometry

Radiometry is the field studying the measurement of electromagnetic radiation in the frequency range between 3×10^{11} and 3×10^{16} Hz, corresponding to wavelengths between 0.01 and 1000 μm. This range includes the ultraviolet, visible and infrared parts of the electromagnetic spectrum. The related field of photometry studies the measurement of electromagnetic radiation perceived by the human vision system; the eye is not equally sensitive to all wavelengths of visible light and photometry accounts for this by taking the spectral response of the eye into consideration when measuring power at each wavelength. The goal of global illumination is to compute the steady-state distribution of light energy in

a scene; this requires an understanding of the physical quantities that represent light energy. Since photometric quantities can be derived from the respective radiometric terms, global illumination algorithms operate on radiometric terms (Dutre *et al.*, 2003).

2.3.1 Radiometric quantities

Light is *radiant energy* and is transported by electromagnetic radiation through space. Radiant energy [Q] is measured in *joules* [J]. Broadband sources, such as the sun, emit electromagnetic radiation throughout the electromagnetic spectrum, as opposed to monochromatic sources, which emit radiation at one specific wavelength. Spectral radiant energy is the amount of radiant energy per unit wavelength interval at wavelength λ. It is measured in *joules per nanometre* $[J \cdot nm^{-1}]$ and is defined as:

$$Q_\lambda = \frac{\mathrm{d}Q}{\mathrm{d}\lambda} \tag{2.10}$$

The human eye perceives colour via specialised cells containing pigments with different spectral sensitivities, known as *cone cells*. There are three types of cones, sensitive to three overlapping frequency ranges. Any colour perception can be represented by integrating energy over these spectra. The *tristimulus theory* of colour perception states that all spectral power distributions can be represented using three scalar values instead of using complete functions. In computer graphics, spectral quantities are usually modelled using a discrete set of wavelengths expressed as a coordinate triple in the CIE XYZ colour space (Smith & Guild, 1931).

Radiant flux [Φ] (also known as radiant power) is the radiant energy passing through a surface of interest per unit time; it is measured in *watts* [W].

$$\Phi(t) = \frac{\mathrm{d}Q}{\mathrm{d}t} \tag{2.11}$$

Radiant flux density is the radiant flux per unit area at a point on a surface. The surface can be real or a mathematical abstraction. Radiant flux density incident to a surface (arriving at a surface), is referred to as *irradiance* [E] and defined as:

$$E(\mathbf{x}) = \frac{\mathrm{d}\Phi(\mathbf{x})}{\mathrm{d}A_x} \tag{2.12}$$

where $\Phi(\mathbf{x})$ is the radiant flux arriving at $\mathbf{x}$ and $\mathrm{d}A_x$ is the differential area

surrounding it. Radiant flux leaving a surface due to reflection or emission is called *radiance exitance* [M] (or *radiosity* [B]) and is defined as:

$$M(\mathbf{x}) = B(\mathbf{x}) = \frac{\mathrm{d}\Phi(\mathbf{x})}{\mathrm{d}A_x} \tag{2.13}$$

where $\Phi(\mathbf{x})$ is the radiant flux leaving $\mathbf{x}$ and $\mathrm{d}A_x$ is the differential area surrounding the point. Radiant flux density is measured in *watts per square metre* $[W \cdot m^{-2}]$.

Radiance [L] is the amount of energy travelling at some point in a specified direction, per unit time, per unit area perpendicular to the direction of travel, per unit solid angle (Sillion *et al.*, 1994). Radiance is measured in *watts per square metre per steradian* $[W \cdot m^{-2} \cdot sr^{-1}]$ and is defined as:

$$L(\mathbf{x}, \omega) = \frac{\mathrm{d}^2\Phi(\mathbf{x}, \omega)\ r^2}{\mathrm{d}A_x \cos\theta_x\ \ \mathrm{d}A_y \cos\theta_y} = \frac{\mathrm{d}^2\Phi(\mathbf{x}, \omega)}{\mathrm{d}A_x\ \ \cos\theta_x\ \ \mathrm{d}\omega}. \tag{2.14}$$

It has the highly desirable property of being spatially invariant along a straight path; for two points, $\mathbf{x}$ and $\mathbf{y}$ which have a line of sight between them, the radiance arriving at $\mathbf{y}$ from the direction of $\mathbf{x}$ is the same as the radiance leaving $\mathbf{y}$ in the direction of $\mathbf{x}$, in the absence of participating media. Using transport notation, where $\mathbf{x} \rightarrow \mathbf{y}$ stands for the direction from point $\mathbf{x}$ to point $\mathbf{y}$, this is expressed as:

$$L(\mathbf{x} \rightarrow \mathbf{y}) = L(\mathbf{y} \leftarrow \mathbf{x}). \tag{2.15}$$

Figure 2.5 shows a surface patch A_x radiating light from a differential element of area $\mathrm{d}A_x$ centred at $\mathbf{x}$. The radiation incident upon A_y can be characterised in terms of radiance; R is a ray segment whose length is the distance between

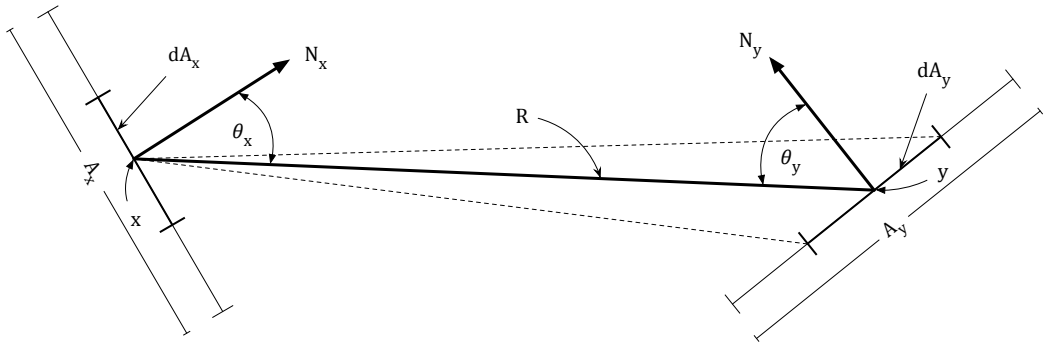


Figure 2.5: Optical radiation passing from one surface to another.

$\mathbf{x}$ and $\mathbf{y}$, and N_x and N_y are the normal vectors to the respective points on the surfaces. From Equation 2.14, the flux $\mathrm{d}^2\,\Phi$ leaving the surface is:

$$\mathrm{d}^2\,\Phi(\mathbf{x},\omega) = L(\mathbf{x}\to\mathbf{y})\ \mathrm{d}A_x\cos\theta_x\,\mathrm{d}\omega \tag{2.16}$$

$$= L(\mathbf{x}\to\mathbf{y})\ \mathrm{d}A_x(N_x\cdot\hat{R})\,\mathrm{d}\omega. \tag{2.17}$$

Irradiance may be expressed in terms of radiance as:

$$E(\mathbf{x}) = \int_\Omega L(\mathbf{x},\omega)\cdot\cos\theta\ \mathrm{d}\omega \tag{2.18}$$

where Ω is the upper hemisphere centred on the surface normal at $\mathbf{x}$; differential irradiance would thus be defined as:

$$\mathrm{d}E(\mathbf{x},\omega) = L(\mathbf{x},\omega)\cdot\cos\theta\,\mathrm{d}\omega. \tag{2.19}$$

Therefore, radiance can be alternatively expressed in terms of differential irradiance:

$$L(\mathbf{x},\omega) = \frac{\mathrm{d}E(\mathbf{x},\omega)}{\cos\theta\,\mathrm{d}\omega}. \tag{2.20}$$

2.4 Bidirectional Scattering Distribution Function

The reflectance properties of a material and the way it interacts with light determine the appearance of an object. Some materials may appear as mirrors, while others, such as chalk, appear as diffuse surfaces. In the general case, light enters a surface at a point $\mathbf{x_i}$ at an angle of incidence ω_i, and leaves at some other point $\mathbf{x_o}$ along some direction ω_o. The relationship between incoming and outgoing radiance is modelled by the *Bidirectional Scattering Surface Reflectance Distribution Function* (BSSRDF), although for the purpose of computer graphics a number of assumptions are commonly made (Dutre *et al.*, 2003):

- Light incident at a surface exits at the same wavelength and at the same time, ignoring effects such as fluorescence and phosphorescence.
- Light incident at some point on a surface, exits at the same point ($\mathbf{x_i}\leftrightarrow\mathbf{x_o}$), precluding subsurface scattering.

The reflectance properties of a surface are described by a function called the *Bidirectional Scattering Distribution Function* (BSDF), defined over the sphere

of directions around a surface point (spanning a solid angle of $4\pi\ sr$). The upper and lower hemispheres of the BSDF are typically separated into two reflectance functions, the *Bidirectional Reflectance Distribution Function* (BRDF) and the *Bidirectional Transmission Distribution Function* (BTDF) respectively (Nicodemus, 1965; Matusik, 2003).

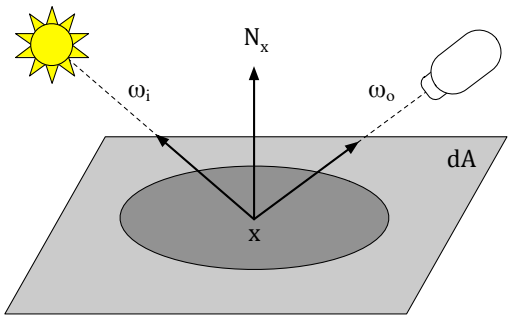


Figure 2.6: Terms in the BRDF.

The BRDF at a point $\mathbf{x}$ is defined as the ratio of the differential radiance reflected towards ω_o and the differential irradiance incident through a differential solid angle $\mathrm{d}\omega_i$:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{\mathrm{d}L(\mathbf{x}, \omega_o)}{\mathrm{d}E(\mathbf{x}, \omega_i)} \tag{2.21}$$

$$= \frac{\mathrm{d}L(\mathbf{x}, \omega_o)}{L(\mathbf{x}, \omega_i)\ \cos\theta_i\, \mathrm{d}\omega_i} \tag{2.22}$$

$$= \frac{\mathrm{d}L(\mathbf{x}, \omega_o)}{L(\mathbf{x}, \omega_i)\ (N_x \cdot \omega_i)\, \mathrm{d}\omega_i}. \tag{2.23}$$

The BRDF (and BTDF) must obey a number of properties in order to satisfy physical correctness, including:

Positivity The function is non-negative:

$$f_r(\mathbf{x}, \omega_i, \omega_o) \geq 0. \tag{2.24}$$

Helmholtz reciprocity Incoming and outgoing light can be considered as reversals of each other; if the incoming and outgoing directions are exchanged, the density remains invariant:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = f_r(\mathbf{x}, \omega_o, \omega_i). \tag{2.25}$$

Energy conservation The total energy in a system cannot change, and thus, the exitant radiance must be less than or equal to the incoming radiance for all possible incident radiance functions; i.e., provided it is not a light source, a surface should never scatter more energy than it receives:

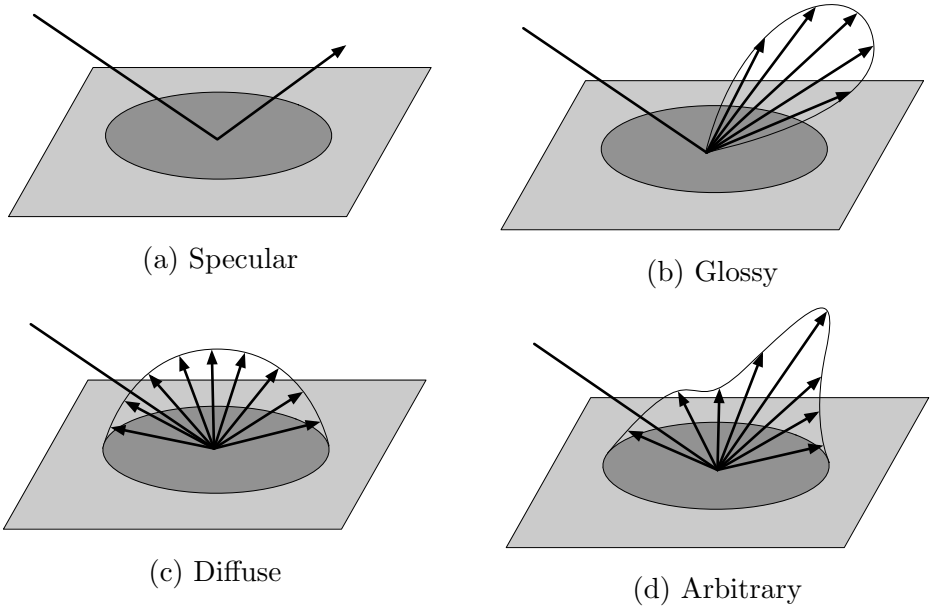
$$\forall\, \mathrm{d}\omega_i, \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o)\ (N_x \cdot \omega_o)\ \ \mathrm{d}\omega_o \leq 1. \tag{2.26}$$

The albedo ρ_α (the proportion of incident light reflected by $\mathbf{x}$) is defined by:

$$\rho_\alpha(\mathbf{x}, \omega_o) = \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o)(N_x \cdot \omega_i)\, \mathrm{d}\omega_i \tag{2.27}$$

2.4.1 Material Models

The appearance of a material is dictated by the nature of its BRDF. In general, materials appear as diffuse, specular or glossy.



(a) Specular (b) Glossy

(c) Diffuse (d) Arbitrary

Figure 2.7: BRDF models for different material appearances.

2.4.2 Diffuse BRDF

Diffuse materials simulate rough surfaces that reflect a portion of incoming light uniformly in all directions. It is defined as (László, 1999):

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{\rho_\alpha(\mathbf{x}, \omega_o)}{\pi} \tag{2.28}$$

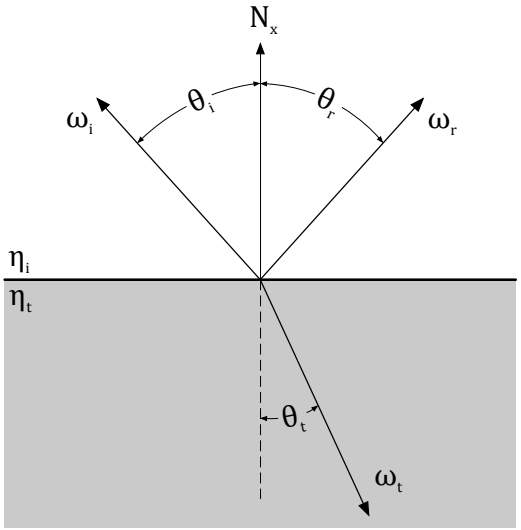


Figure 2.8: Geometry of reflection and refraction.

2.4.3 Specular BRDF

An ideal specular surface obeys the reflection law of geometric optics (Figure 2.8); light from an incoming direction ω_i is reflected along an outgoing direction ω_r. The angles θ_i and θ_r, which vectors ω_i and ω_r make with respect to the surface normal N are equal ($\theta_i = \theta_r$). N, ω_i and ω_r all lie on the same plane. Such surfaces reflect light only along an ideal reflection vector ω_r, which is given by:

$$\omega_r = Re(N, \omega_i) = 2(N \cdot \omega_i)N - \omega_i. \tag{2.29}$$

Thus, for an ideal specular surface the probability of a photon being reflected along a vector ω_r where $\theta_r \neq \theta_i$ is zero $[P(\omega_r|\theta_i \neq \theta_r) = 0]$, and conversely, one when $\theta_r = \theta_i$ $[P(\omega_r|\theta_i = \theta_r) = 1]$. This probability distribution can be modelled using a Dirac delta function: $\delta(\omega_o - Re(N, \omega_i))$. Snell's law (Equation 2.30) is

used to determine refraction in specular transmissive surfaces.

$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2. \tag{2.30}$$

The direction of transmission, ω_t, is given by:

$$\omega_t = T(N, \omega_i) = -\frac{\eta_i}{\eta_t}\omega_i + N\left[\frac{\eta_i}{\eta_t}\cos\theta_i - \sqrt{1 - \left(\frac{\eta_i}{\eta_t}\right)^2 (1 - \cos^2\theta_i)}\right] \tag{2.31}$$

$$= -\frac{\eta_i}{\eta_t}\omega_i + N\left[\frac{\eta_i}{\eta_t}(N \cdot \omega_i) - \sqrt{1 - \left(\frac{\eta_i}{\eta_t}\right)^2 \left(1 - (N \cdot \omega_i)^2\right)}\right] \tag{2.32}$$

The Fresnel equations provide a basis for computing the amount of light that is reflected or refracted at a material interface. The computed reflectance depends on the polarisation of incident light. Two cases of polarisation are considered: *s-polarised* light, where incident light is polarised with its electric field perpendicular to the plane containing the incident reflected and refracted light rays, and *p-polarised* light, where the electric field is parallel to the aforementioned plane. The reflectance for s-polarised light is defined as:

$$F_r^{\perp} = \left\| \frac{\eta_i \cos\theta_i - \eta_t \cos\theta_t}{\eta_i \cos\theta_i + \eta_t \cos\theta_t} \right\|^2, \tag{2.33}$$

while the reflectance for p-polarised light is defined as:

$$F_r^{\parallel} = \left\| \frac{\eta_i \cos\theta_t - \eta_t \cos\theta_i}{\eta_i \cos\theta_t + \eta_t \cos\theta_i} \right\|^2. \tag{2.34}$$

If polarisation is not taken into account, that is, incident light is assumed to contain an equal mix of both polarisations, the Fresnel reflection coefficient is:

$$F_r = \frac{F_r^{\perp} + F_r^{\parallel}}{2}. \tag{2.35}$$

As a consequence of energy conservation, the Fresnel transmission coefficient F_t is given as:

$$F_t = 1 - F_r. \tag{2.36}$$

The BRDF for specular reflection is thus defined as:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = F_r \cdot \delta(\omega_o - R(N, \omega_i)) \tag{2.37}$$

while the BTDF for specular transmission is given by:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{\eta_t^2}{\eta_i^2} F_t \cdot \delta(\omega_o - T(N, \omega_i)) \tag{2.38}$$

2.4.4 Glossy BRDF

Many real world materials are neither perfectly diffuse nor ideal specular reflectors; glossy BRDFs model the spectrum of surface types that lie within these two extrema. The BRDF of glossy materials is often difficult to model analytically (Dutre *et al.*, 2003); however, given their applicability, they have been extensively researched.

The *microfacet* model was introduced to computer graphics by Cook & Torrance (1982) to model the reflection of light from rough surfaces, and was based on the work of Torrance & Sparrow (1967) in the field of optics. Microfacets model a collection of tiny smooth mirrored planar surfaces, known as a microsurface, whose detail is too small to be seen directly. The model replaces the microsurface by a simplified surface with a modified scattering function that matches the aggregate directional scattering for the collection of microfacets. Microsurfaces are represented by two statistical measures, the distribution of surface normals and a shadow masking function which models the self-shadowing of the microfacets. Some models, such as Schlick (1994) and Ashikhmin & Shirley (2000), also take Fresnel effects into consideration.

Ward (1992) introduced a simplification of the microfacet model that also accounts for surfaces which exhibit anisotropy. The model is known as a separable model because it accounts for diffuse and glossy components separately:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{\rho_d}{\pi} + \frac{\rho_s}{4\pi\alpha_x\alpha_y\sqrt{\cos\theta_i \cos\theta_o}}\, e^{-tan^2\theta_h\left(\frac{\cos^2\phi_h}{\alpha_x^2}+\frac{\sin^2\phi_h}{\alpha_y^2}\right)} \tag{2.39}$$

where ρ_s controls the magnitude of the lobe, and α_x and α_y control the width of the lobe in the two principal directions of anisotropy. (θ_h, ϕ_h) are the angles between the surface normal and the half angle (Walter, 2005).

Lewis (1994) proposed the Modified Phong BRDF (also known as the classical cosine lobe model), a physically-correct version of the Phong BRDF (Phong, 1975). Similarly to Ward (1992), this model is separable, and is composed of a diffuse lobe and a glossy lobe represented by a cosine lobe oriented with the axis of the surface normal.

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{\rho_d}{\pi} + \rho_s C_s \cos^n \alpha, \tag{2.40}$$

where ρ_s is the maximum albedo, C_s is a normalisation factor ($C_s = (n+2)/2\pi$) such that $0 \leq \rho_s \leq 1$, n is the strength of the highlight, and α is the angle between ω_o and the reflection of ω_i about the surface normal N. This can be rewritten as:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{\rho_d}{\pi} + \rho_s C_s \left[Re(N, \omega_i) \cdot \omega_o\right]^n \tag{2.41}$$

Lafortune *et al.* (1997) is also a separable BRDF made of a mixture distribution of oriented cosine lobes. This is given by:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{\rho_d}{\pi} + \rho_s C_s \sum_{j=1}^{N_{lobes}} \left[Re(W_j, \omega_i) \cdot \omega_o\right]^{n_j} \tag{2.42}$$

where N_{lobes} is the number of lobes, W_j is the axis about which the j^{th} lobe is oriented, and n_j is the exponent of the j^{th} lobe.

2.5 Light Transport

High-fidelity rendering makes use of physically-based quantities to compute light transport through a virtual scene, simulating the interactions between light and surfaces in order to compute energy levels in the scene.

2.5.1 The Rendering Equation over The Hemisphere

The equilibrium of light energy can be formulated mathematically via an integral equation known as the Rendering Equation (Kajiya, 1986), which has inspired a large variety of rendering algorithms and provides a definition of radiance at any

point in the scene:

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega f_r(\mathbf{x}, \omega_o, \omega_i)\ L_i(\mathbf{x}, \omega_i)\ \cos\theta_i\ \mathrm{d}\omega_i \tag{2.43}$$

$$= L_e(\mathbf{x}, \omega_o) + \int_\Omega f_r(\mathbf{x}, \omega_o, \omega_i)\ L_i(\mathbf{x}, \omega_i)\ (N_x \cdot \omega_i)\ \mathrm{d}\omega_i \tag{2.44}$$

where $\mathbf{x}$ is a point in the scene, L_e is the emitted light, ω_i and ω_o are the incoming and outgoing directions of light respectively, f_r is the BRDF and $\cos\theta_i$ is the foreshortening factor.

2.5.2 Area Formulation

The hemispherical formulation of the rendering equation (Equation 2.44) can be alternatively formulated in terms of the surfaces of objects in a scene that contribute to the incoming radiance at the point $\mathbf{x}$ (Dutre *et al.*, 2003):

$$L_o(\mathbf{x}' \to \mathbf{x}) = L_e(\mathbf{x}' \to \mathbf{x}) + \int_S f_r(\mathbf{x}'' \to \mathbf{x}' \to \mathbf{x})\ L_i(\mathbf{x}'' \to \mathbf{x})\ G(\mathbf{x}' \to \mathbf{x})\, \mathrm{d}s \tag{2.45}$$

where the domain of integration S specifies all the surfaces in the scene. The BRDF (f_r) is expressed in three point notation (light incoming at $\mathbf{x}'$ from the direction of $\mathbf{x}''$ and reflected towards $\mathbf{x}$). $G(\mathbf{x} \to \mathbf{y})$ is known as the *geometry term* and is defined as:

$$G(\mathbf{x} \to \mathbf{y}) = V(\mathbf{x} \to \mathbf{y}) \frac{\cos\theta_x \cos\theta_y}{|\mathbf{y} - \mathbf{x}|^2} \tag{2.46}$$

$$= V(\mathbf{x} \to \mathbf{y}) \frac{|N_x \cdot \omega_i|\ \ |N_y \cdot -\omega_i|}{|\mathbf{y} - \mathbf{x}|^2} \tag{2.47}$$

and $V(\mathbf{x} \to \mathbf{y})$ is the *visibility term* determining whether points $\mathbf{x}$ and $\mathbf{y}$ are within the line of sight of each other (unoccluded):

$$V(\mathbf{x} \to \mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ and } \mathbf{y} \text{ unoccluded} \\ 0 & \text{otherwise} \end{cases} \tag{2.48}$$

The geometry term follows from the definition of the solid angle in Equation 2.3, but also includes the cosine term (θ_i) from the rendering equation. The visibility term can be expressed in terms of a ray casting operator $R(\mathbf{x}, \omega)$ which

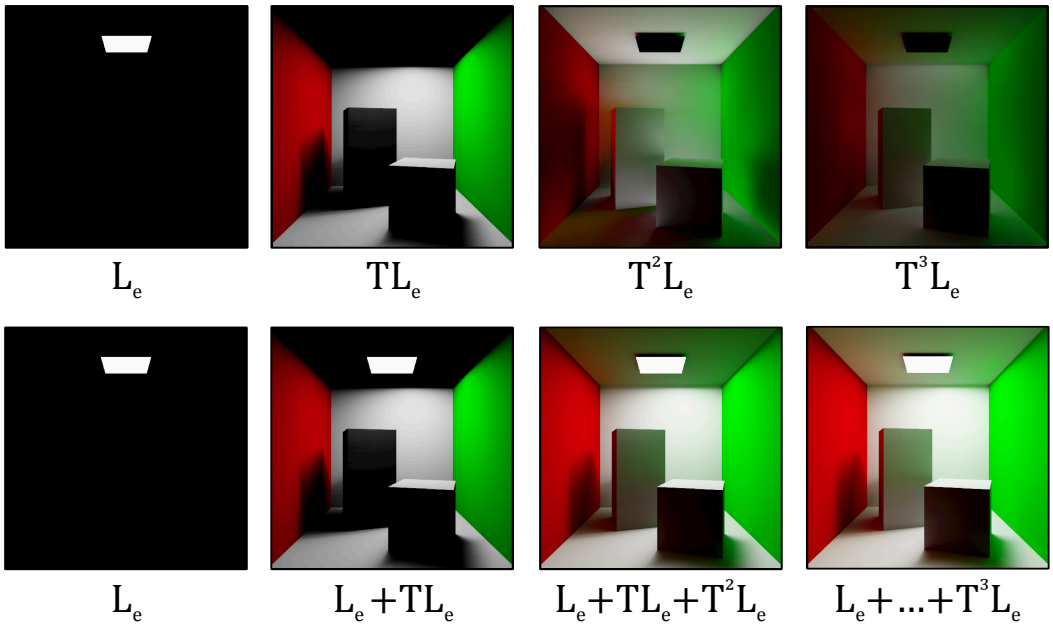


Figure 2.9: Visualisation of the terms of the Transport formulation for the Rendering Equation.

gives the scalar distance from $\mathbf{x}$ to the closest surface along the direction ω

$$V(\mathbf{x} \to \mathbf{y}) = \begin{cases} 1 & R(\mathbf{x}, \omega) \geq |\mathbf{y} - \mathbf{x}| \\ 0 & \text{otherwise} \end{cases} \tag{2.49}$$

where $\omega = \dfrac{\mathbf{y} - \mathbf{x}}{|\mathbf{y} - \mathbf{x}|}$.

2.5.3 Transport Formulation

The rendering equation is a Fredholm Integral Equation of the second kind, the canonical form of which is expressed as follows:

$$\phi(t) = f(t) + \int K(t, s)\ \phi(s)\ \mathrm{d}s. \tag{2.50}$$

$K(t, s)$ is known as the kernel of the integral. The integral equation may be expressed in terms of a linear operator T such that:

$$(T \circ \phi)(t) = \int K(t, s)\ \phi(s)\ \mathrm{d}s, \tag{2.51}$$

and thus, the rendering equation may be rewritten in linear operator form as follows:

$$L = L_e + TL \tag{2.52}$$
$$IL - TL = L_e \tag{2.53}$$
$$(I - T)L = L_e \tag{2.54}$$
$$L = (I - T)^{-1} L_e \tag{2.55}$$

where I is the identity operator. Using a Neumann expansion, Equation 2.55 may be expanded as follows:

$$L = (I - T)^{-1} L_e = L_e \sum_{n=0}^{\infty} T^n \tag{2.56}$$
$$= L_e(I + T + T^2 + T^3 + \ldots) \tag{2.57}$$
$$= L_e + TL_e + T^2 L_e + T^3 L_e + \ldots \tag{2.58}$$

This formulation of the rendering equation is also known as the transport formulation. The individual terms and their aggregation are visualised in Figure 2.9.

2.6 Monte Carlo Methods

Solving the rendering equation requires evaluating high-dimensional integrals. Numerical quadrature methods evaluate the integrand for a finite number of samples from within the domain and generate the integral as a weighted sum of these values. A general form of the solution is (László, 1999):

$$I = \int_V f(x)\,\mathrm{d}x \approx \sum_{i=1}^{N} f(x_i) \cdot w(x_i) \tag{2.59}$$

where x_i is a sample point from the domain V, and w is a weighting function of the given quadrature rule. The required number of sample points to find an estimate with error ϵ in an s-dimensional space is proportional to $(\Delta f/\epsilon)^s$, for the midpoint rule, making the computational complexity exponential with regards to the dimension of the domain. This phenomenon is known as the *curse of dimensionality* and is also exhibited by other classical quadrature rules such

as the trapezoidal or Simspon's Rule. The curse of dimensionality can be avoided by the use of Monte Carlo or quasi-Monte Carlo integration methods (Metropolis & Ulam, 1949; Caflisch, 1998).

2.6.1 Monte Carlo Integration

The principle behind Monte Carlo integration is that of drawing random samples from the integrand and averaging them to provide an estimate of the integral. For a function $f(x)$ whose integral is $I = \int f(x)\,\mathrm{d}x$, the Monte Carlo estimator $< I >$ for N samples and probability distribution function $p(x)$ is given by:

$$< I >= \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}. \tag{2.60}$$

In the limit, as $N \to \infty$, $< I > \to I$. The estimator $< I >$ converges at a rate $\mathrm{O}\big(N^{-1/2}\big)$ and is independent of dimension. In comparison, grid-based quadratures converge at a rate $\mathrm{O}\big(N^{-k/s}\big)$ for an order k method in s-dimensions. Monte Carlo methods make it possible to solve the rendering equation (see Equation 2.44) by generating N samples over the hemisphere of directions Ω:

$$< L_o(\mathbf{x}, \omega_o) >= L_e(\mathbf{x}, \omega_o) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\mathbf{x}, \omega_o, \omega_i)\ L_i(\mathbf{x}, \omega_i)\ (N_x \cdot \omega_i)}{p(\omega_i)}. \tag{2.61}$$

2.7 Sampling

In Equation 2.61, samples are drawn from a probability distribution $p(\omega_i)$, where ω_i is a vector in the hemisphere of directions Ω. There are various strategies for choosing ω_i, such as uniform random sampling or stratified sampling. The latter divides the domain of the distribution into equally-sized subdomains or strata and randomly samples within the boundaries of each stratum. Stratified sampling generally contributes towards lowering variance, although in the worst case, it is equivalent to random sampling.

Another sampling approach is importance sampling, where ω_i is chosen such that $p(\omega_i)$ decreases variance in the solution. Thus, ω_i is chosen more frequently where the integrand is large or more *important*, and consequently, contributes more to the solution. A caveat is that the probability distribution $p(\omega_i)$ should have a similar shape as the function being integrated.

2.7.1 Low-discrepancy (Quasi-random) Sampling

An approach that also minimises variance by reducing the clumping of samples is that of low-discrepancy sampling. Intuitively, low-discrepancy sequences are deterministic sequences that try to minimise variability in the density of the points from one location to the next (Jarosz, 2008). A formal definition of discrepancy is given in Pharr & Humphreys (2010).

There are a number of techniques that are used to generate low-discrepancy sequences. The radical inverse function is used to express an integer (in a base b) in the range $[0, 1)$, and can be used to construct a number of low-discrepancy sequences (Van der Corput, 1936; Halton, 1964). Particularly, for an integer i:

$$i = \sum_{j=0}^{\infty} a_j b^j, \tag{2.62}$$

where $a_j \in \{0, \ldots, b-1\}$, the radical inverse function $\Phi_b(i)$ is given by:

$$\Phi_b(i) = \sum_{j=0}^{\infty} a_j b^{-j-1} \tag{2.63}$$

The van der Corput low-discrepancy sequence (Van der Corput, 1936) given by:

$$x_i = \Phi_b(i) \tag{2.64}$$

is the one-dimensional generalisation of the Halton sequence (Halton, 1964). The s-dimensional Halton sequence is defined as:

$$x_i = (\Phi_{b_1}(i), \Phi_{b_2}(i), \Phi_{b_3}(i), \ldots, \Phi_{b_s}(i)), \tag{2.65}$$

with $b_1, b_2, b_3, \ldots, b_s$ being *relative primes*. Two integers a and b are said to be relative primes if their highest common factor is 1. Numerous other low-discrepancy sequences have been proposed; for more details see (Hammersley, 1960), (Faure, 1982), (Sobol, 1967) and Niederreiter (1988).

2.8 Spectra

Spectral power distributions can be modelled using a discrete set of wavelengths expressed as an triple in the CIE XYZ colour space. The Y component has been defined as luminance, while the XZ components are defined as the plane of chromaticities at the given luminance. Display devices based on the RGB system require a transformation of spectral power from the XYZ colour space prior to visualisation. This is defined as:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 0.41847 & -0.15866 & -0.082835 \\ -0.091169 & 0.25243 & 0.015708 \\ 0.0009209 & -0.0025498 & 0.1786 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.66}$$

and the respective inverse transformation is given by:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{0.17697} \begin{bmatrix} 0.49 & 0.31 & 0.20 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.00 & 0.01 & 0.99 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{2.67}$$

The RGB triples for low dynamic range display devices are usually in the range of $[0 \ldots 255]$, with each channel using 8-bits to store the respective contribution. The limited dynamic range offered by such devices fails to reproduce the greater range of luminance levels found in natural scenes. High-fidelity rendering simulates this higher dynamic range of luminance levels and uses *tone mapping* to obtain a perceptual match on devices that cannot reproduce the full range, while trying to retain some characteristics of the original signal such as local and global contrast (Banterle *et al.*, 2011).

In order to manipulate colour triples, a number of operations are defined. Specifically, the addition of two colour triples A and B is defined as:

$$A + B \stackrel{def}{=} \begin{bmatrix} A_r + B_r & A_g + B_g & A_b + B_b \end{bmatrix}^T \tag{2.68}$$

The multiplication of a colour triple A by a scalar c:

$$cA \stackrel{def}{=} \begin{bmatrix} cA_r & cA_g & cA_b \end{bmatrix}^T \tag{2.69}$$

while the component-wise multiplication of two colour triples is defined as:

$$A \oplus B \stackrel{def}{=} \left[\begin{array}{ccc} A_r B_r & A_g B_g & A_b B_b \end{array} \right]^T \tag{2.70}$$

Frequently, there is a need for manipulating multiple colour triples which are logically organised in an $m \times n$ grid configuration. These are representative of synthesised raster images, texture maps and other bitmap structures containing colour information:

$$P = \left[\begin{array}{cccc} P_{11} & P_{12} & \cdots & P_{1n} \\ P_{21} & P_{22} & \cdots & P_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{m1} & P_{m2} & \cdots & P_{mn} \end{array} \right] \tag{2.71}$$

and are henceforth referred to simply as *images*. The addition of two $m \times n$ images P and Q is defined as:

$$P + Q \stackrel{def}{=} \left[\begin{array}{cccc} P_{11} + Q_{11} & P_{12} + Q_{12} & \cdots & P_{1n} + Q_{1n} \\ P_{21} + Q_{21} & P_{22} + Q_{22} & \cdots & P_{2n} + Q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{m1} + Q_{m1} & P_{m2} + Q_{m2} & \cdots & P_{mn} + Q_{mn} \end{array} \right], \tag{2.72}$$

while their component-wise multiplication is given by:

$$P \oplus Q \stackrel{def}{=} \left[\begin{array}{cccc} P_{11} \oplus Q_{11} & P_{12} \oplus Q_{12} & \cdots & P_{1n} \oplus Q_{1n} \\ P_{21} \oplus Q_{21} & P_{22} \oplus Q_{22} & \cdots & P_{2n} \oplus Q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{m1} \oplus Q_{m1} & P_{m2} \oplus Q_{m2} & \cdots & P_{mn} \oplus Q_{mn} \end{array} \right]. \tag{2.73}$$

The multiplication of an $m \times n$ image P by a scalar c is defined as:

$$cP \stackrel{def}{=} \left[\begin{array}{cccc} cP_{11} & cP_{12} & \cdots & cP_{1n} \\ cP_{21} & cP_{22} & \cdots & cP_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ cP_{m1} & cP_{m2} & \cdots & cP_{mn} \end{array} \right]. \tag{2.74}$$

2.9 Summary

This chapter has provided a background to many fundamental concepts related to high-fidelity rendering. First, a brief introduction to solid angles and Radiometry was provided. The bidirectional reflectance distribution function was then discussed, together with respective physically-based models for diffuse, specular and glossy surfaces. This was followed by the rendering equation in three of its formulations: hemisphere, area and transport. A brief discussion of Monte Carlo methods and Monte Carlo integration followed, providing practical ways of solving the equation despite its high dimensionality. Next, sampling approaches were briefly mentioned, underscoring their importance in reducing variance within Monte Carlo integration, before providing a representation of spectral power distributions and operators for their manipulation. This background is provided as a basis for the high-fidelity rendering techniques described in the next chapter.

CHAPTER 3

High-fidelity rendering

High-fidelity rendering is a term denoting image synthesis through the use of physically-based lighting models and perceptually-based rendering procedures, with the aim of generating images that are visually indistinguishable from real-world images (Greenberg *et al.*, 1997). The primary concern of high-fidelity rendering is the accuracy and fidelity of the physical simulation. Image synthesis algorithms are generally classified into two broad categories: *object-order* and *image-order* (Shirley *et al.*, 2009). Object-order methods perform per-object operations during synthesis to find the image plane pixels that are influenced by the given object, while image-order methods perform per-pixel operations on the image plane to find the scene objects that influence the given pixel. Typical high-fidelity rendering methods use an image-order approach, with solutions provided by point-based *ray tracing methods*. The object-order approach is generally preferred in real-time rendering scenarios due to its suitability for use on Graphics Processing Units (GPUs).

3.1 High-fidelity Image Synthesis Framework

Greenberg *et al.* (1997) proposed a three-stage system as a general and comprehensive framework for physically-based rendering. The first two stages simulate physical processes, whereas the third is concerned with the perceptual issues tied with displaying the output of the previous stages:

Light reflection models are concerned with the acquisition and development of models for arbitrary reflectance functions and their efficient representations. These models are validated experimentally.

Light transport simulation specifies the creation of global illumination algorithms that can accurately simulate light energy transport within complex environments and their validation through comparisons to measured physical environments.

Perceptual issues deal with mapping the simulated radiance quantities to a display device, taking into account the physical characteristics of the device as well as the conditions under which the produced image will be viewed.

The focus of this thesis within the remit of the framework specified by Greenberg *et al.* (1997) is light transport simulation, particularly the creation of global illumination algorithms and their efficient representation and implementation via the use of distributed computing. Perceptual issues are tangentially skirted during the mapping of physically-correct radiance values to the ranges supported by common display devices; this is accomplished through tone mapping algorithms. In this work, Schlick (1995), Durand & Dorsey (2002) and Drago *et al.* (2003) are principally used.

A high-level approach to image synthesis is given in Algorithm 1. Broadly, the steps in the algorithm entail initialising the scene or virtual environment by making assets such as geometry and materials accessible to the rendering algorithms (line 2). The rendering loop (lines 3-7) consists of animating the scene, simulating light transport and presenting the results of each frame; animation may respond to feedback interactively, in the case of real-time rendering applications such as video games, or follow scripted sequences, for offline applications such as movie production rendering.

Algorithm 1 Generic rendering pipeline for image and object-order approaches.

```
1: procedure RenderingPipeline(scene, eye)
2:     Prepare(scene)
3:     for each frame do
4:         Animate(scene)
5:         frame ← Render(scene, eye)
6:         Present(frame)
7:     end for
8: end procedure
```

The image and object-order approaches have been abstracted by the `Render` function, which is the exclusive focus of further discussion in this chapter.

3.2 Image-order Approach

In order to synthesise a picture using an image-order approach, a weighted integral over the image plane of incident radiance values must be computed. These radiance values are defined along rays coming from the scene and pointing to the eye, passing through the individual pixels of the image plane. This is described by Dutre *et al.* (2003) as:

$$L_{pixel} = \int_{imageplane} L(\mathbf{p} \to \mathbf{e}) H(\mathbf{p}) \, \mathrm{d}\mathbf{p} \tag{3.1}$$

$$= \int_{imageplane} L(\mathbf{x} \to \mathbf{e}) H(\mathbf{p}) \, \mathrm{d}\mathbf{p}, \tag{3.2}$$

where $\mathbf{p}$ is a point on the image plane, $H(\mathbf{p})$ a weighting or filtering function, and $\mathbf{x}$ is the visible point seen from the eye ($\mathbf{e}$) through $\mathbf{p}$. Algorithm 2 illustrates how an image would be synthesised by computing the radiance values for a number of rays starting from the eye, intersecting the image plane and traversing the scene. The rays are chosen such that the weight of their contribution is non-zero (line 5: $H(\mathbf{p}) \neq 0$). The `Radiance` function evaluates the rendering equation to estimate the radiance along *ray* (see §2.5.1).

Algorithm 2 Image-order approach: the Ray Tracing algorithm.

```
 1: procedure RENDER(scene, eye)
 2:     for each pixel ∈ imagePlane do
 3:         radiance ← 0
 4:         for each ray ∈ viewingRays do
 5:             pick sample point p such that H(p) ≠ 0
 6:             construct ray at origin e in direction p − e
 7:             radiance ← radiance + Radiance(ray) ∗ H(p)
 8:         end for
 9:         radiance ← radiance/|viewingRays|
10:     end for
11: end procedure
```

Heckbert (1990) introduced a general notation for describing the interactions along a photon's path from light (L) to eye (E), characterising these interactions as either diffuse (D) or specular (S). A formal language specified by the regular expression $L(D|S)^*E$ over the alphabet $\Sigma = \{L, D, S, E\}$ is used to label each path by some string in the language. The diffuse interactions (D) denote any non-specular interactions. However, it is at times useful to differentiate between

diffuse and glossy interactions; thus, the alphabet Σ is extended to include the glossy interaction (G), such that $\Sigma = \{L, D, G, S, E\}$, with the regular expression changing to $L(D|G|S)^*E$. The rules for the construction of these regular expressions make use of a number of operators; the parentheses () are used to group elements, and define their scope. The vertical bar (|) carries the semantics of the boolean *or* and is used to denote alternatives. A number of quantification operators are also employed to specify how often a preceding element is allowed to occur. The Kleene operator (*) means zero or more instances, while the Kleene plus ($^+$) means one or more instances of the preceding element, be it a group or single token. The question mark (?) indicates zero or one instances of the preceding element.

3.2.1 Ray-casting Operator

Ray tracing methods are based on point sampling of the scene being visualised. In particular, the fundamental primitive in ray tracing is the ray-casting operator, which fulfils two main purposes: (1) to determine the closest visible point from another point $\mathbf{x}$ along a direction ω_x and, (2) to determine whether two points $\mathbf{x}$ and $\mathbf{y}$ are mutually visible. In either case, the asymptotic time complexity of naïve ray-casting grows linearly in the number of primitives. It is thus $\mathrm{O}(n)$, where n is the number of primitives in the scene. Furthermore, light transport simulation using ray tracing may spend as much as 75% of its time performing ray-casting and intersection tests (Whitted, 1980). Spatial subdivision algorithms and data structures that are designed to exploit spatial coherence of primitives during search and traversal operations can substantially reduce the complexity of ray-casting. These data structures are called acceleration structures; by and large, there are two classes of subdivision processes that generate these acceleration structures: those that partition objects, resulting in possibly overlapping subspaces, and those that partition space, resulting in disjoint subspaces. Acceleration structures are used to store primitives such as triangles, spheres or point samples amongst others. Reinhard *et al.* (1998) state that as a rule of thumb, the number of cells in a spatially subdivided structure should be of the same order as the number of primitives n in the scene, and that the time complexity of ray-casting using these algorithms is reduced to $\mathrm{O}\left(\sqrt[3]{n}\right)$. A number of subdivision strategies are outlined below.

Grids

Grid structures that subdivide space into regular cubic regions of space are called regular grids. Each region is called a cell, or voxel, and references all the primitives that overlap it. Efficient traversal of the structure can be accomplished using algorithms such as 3D-DDA by Fujimoto *et al.* (1986). In general, although traversal is slower than other acceleration structures, grids have very fast construction times.

Octree

The octree is a hierarchical data structure that subdivides space recursively in eight non-overlapping voxels of equal volume, proposed by Glassner (1988). The recursion process stops when either of two base cases is satisfied: (1) the number of primitives in a voxel is beneath a specified threshold, beyond which no further subdivision is required, or (2) a maximum depth of recursion has been reached. A single-reference octree references a primitive only at a single top-most node that can contain it. Conversely, a multiple-reference octree stores references to a primitive in all nodes it overlaps. The search procedure in multiple-reference octrees is faster than single-reference, since backtracking during tree traversal is eliminated. This comes at the cost of a higher memory footprint (Krivanek & Gautron, 2009).

Kd-tree

Another tree-based acceleration structure is the kd-tree. Space is recursively subdivided into two half-spaces by a partitioning plane, perpendicular to one of the three coordinate axes. There are various heuristics to determine the partitioning plane, or split, such as median-split or the surface area heuristic (SAH) by MacDonald & Booth (1990). Primitives are added to either half-space, depending on which side of the splitting plane they fall; if any primitive straddles the partitioning plane, it is split into smaller primitives which are added to their respective half-spaces. Alternatively, a reference to the primitive can be added to each half-space. The kd-tree is considered to be very fast to search, although slower to build than other acceleration structures.

Bounding Volume Hierarchies

The Bounding Volume Hierarchies (BVH) algorithm proposed by Kay & Kajiya (1986) creates a hierarchy of primitives via the recursive partitioning of the primitives themselves. The two resulting primitive sets at each split are bounded with a tight-fitting volume and assigned to either child of the current node; the internal nodes are further partitioned until each leaf of the tree contains exactly one primitive. The partitioning strategy, similarly to kd-trees, employs methods such as SAH to determine the best splitting point. The BVH is fast to search, although usually not as fast as the kd-tree. The construction, on the other hand, is faster than the kd-tree and thus, it is often used with dynamic scenes, where the primitive configurations change frequently, requiring their acceleration structure to be rebuilt.

3.3 Object-order approach

The object-order approach is dominated by the rasterisation class of rendering algorithms. Rasterisation is the process by which a three dimensional representation of geometry is projected onto a two dimensional plane to synthesise an image. The rasterisation process is often referred to as the real-time rendering pipeline, which consists of three conceptual stages: Application, Geometry and Rasterisation (Akenine-Moller *et al.*, 2008). These stages are further subdivided into other pipeline stages, which are more closely tied to the implementation of functions within the remit of each of the three conceptual stages. The speed of execution of the graphics pipeline, that is, the frequency at which the images are updated, is dependent on the slowest stage in the pipeline. This update rate is expressed in frames per second (fps) or alternatively in Hertz (Hz).

The application stage determines the geometry that is fed to the geometry stage for rendering. This geometry is referred to as rendering primitives and usually consists of points, lines or triangles. The geometry stage performs a number of per-polygon and per-vertex operations, but broadly, the model and view transform stage transforms objects into world space and then into view space. Subsequently, primitives are projected into a cube with extents of $(-1, -1, -1)$ and $(1, 1, 1)$, known as the canonical view volume. Primitives outside the view volume are discarded, while those that lie partially inside are clipped against it. The primitives that have not been discarded are mapped to the screen coordinates.

The output of the geometry stage becomes the input to the rasterisation stage, where the colours of pixels are computed to perform the final stages in image synthesis. In particular, the vertices of the geometry from the previous stage, together with shading information, are used by the scan conversion (or rasterisation) process to generate the output image. An important stage during scan conversion is that of fragment processing, which adds surface details such as texturing (Blinn & Newell, 1976) and lighting to rendered pixels, and performs hidden surface removal using the z-buffer algorithm (Catmull, 1974).

Algorithm 3 Object-order approach: the rasterisation algorithm.

```
1: procedure RENDER(scene, eye)
2:     for each object ∈ scene do
3:         triangles ← ∅
4:         primitives ← GetPrimitives(object)
5:         for each p ∈ primitives do
6:             p_t ← ModelViewTransform(p, eye)
7:             p_p ← Project(p_t)
8:             if view volume contains p_p then
9:                 p_c ← Clip(p_p)
10:                p_n ← Map(p_c)
11:                triangles ← triangles ∩ p_n
12:            end if
13:        end for
14:        for each t ∈ triangles do
15:            for each fragment ∈ t do
16:                if fragment is visible then
17:                    ShadeAndMerge(fragment)
18:                end if
19:            end for
20:        end for
21:    end for
22: end procedure
```

Algorithm 3 illustrates the rasterisation process, which is still the de facto standard in real-time interactive rendering. Non local phenomena, such as shadows, reflection and refraction, require distinct rendering techniques to simulate. In rasterisation, shadows are typically simulated using shadow mapping (Williams, 1978). In a first pass, a shadow map is created by rendering the scene from the point of view of the light source; each pixel in the shadow map contains the distance of that particular point from the light source. This is also known as the depth of the pixel in the shadow map. In a second pass, every rasterised point

that is influenced by the light source is tested against the respective depth value in the shadow map; if found to be larger, the point is in shadow. Akenine-Moller *et al.* (2008) gives a treatise of such algorithms for rasterisation. A very marked advantage of rasterisation over other methods lies in its efficiency. The advent of GPUs, which accelerate rasterisation, have furthered this advantage to the point where consumer hardware can easily render billions of triangles per second. The programmable hardware afforded by modern GPUs has transformed the graphics pipeline, and real-time rendering, via the introduction of complex per-triangle, per-vertex and per-fragment programs, called shaders, that move shading models beyond traditional empirical methods (Gouraud, 1971; Phong, 1975), making real-time physically-based shading a possibility.

3.4 Methods Based on the Ray Tracing Approach

The first to use the ray tracing approach in computer graphics was Appel (1968), who applied ray-casting to determine visible surfaces as well as shadow casting from arbitrarily located light sources (see Figure 3.1). In this approach, light is not propagated to other surfaces beyond the primary intersection and thus computes LDE paths in the notation introduced earlier. Light incident on a surface interacts with the surface such that it may be absorbed, reflected or transmitted, depending on the material properties of the surface. The approach introduced by Whitted (1980) simulates these interactions by propagating light after the first intersection for specular and transmissive surfaces, although paths are still terminated if they encounter a diffuse or glossy surface (see Figure 3.2). The paths generated by Whitted (1980) are expressed by $L(D|G)?S^*E$.

3.4.1 Distributed Ray Tracing

Cook *et al.* (1984) introduced a new approach, termed distributed ray tracing, which performed oversampling of rays in space and distributed them over time (see Figure 3.3). The method prefaced stochastic ray tracing methods, which used Monte Carlo techniques to sample complex BRDFs and model many high-fidelity aspects such as glossy reflections and transmissions (blurred transparency), produce penumbras from area light sources (as opposed to Whitted (1980), which supported only point lights), simulate depth-of-field and generate motion blur from moving objects. Similarly to Whitted (1980), rays are recursively traced

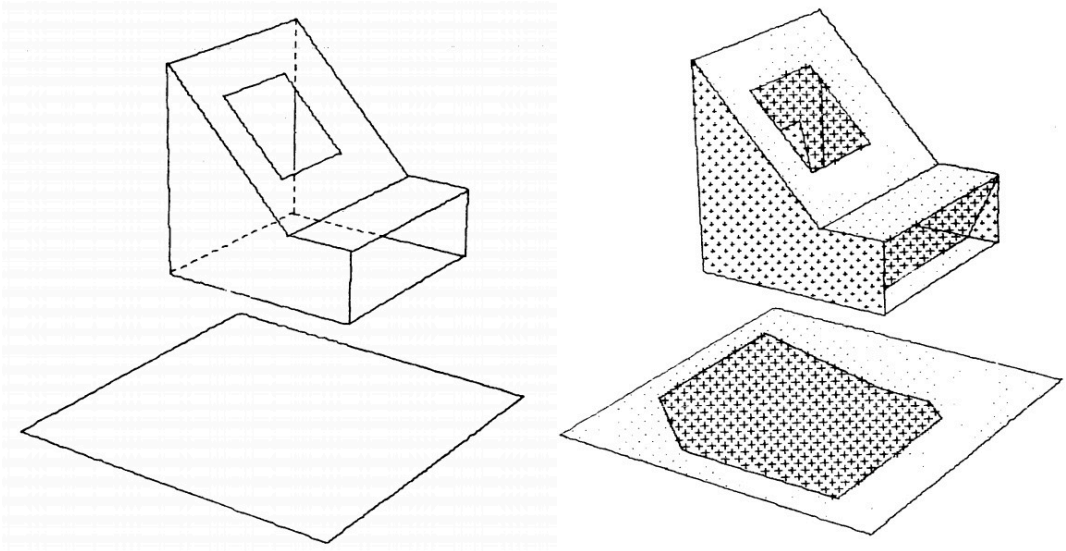


Figure 3.1: A machine-generated line drawing and the same line drawing shaded using Appel's method $[L(D|G)E]$ (Appel, 1968).

through the scene; at each point of intersection, a set of rays is generated over the hemisphere via stochastic sampling. For a point $\mathbf{x}$, this can be formally expressed as:

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) |\cos \theta_i|}{p(\omega_i)}, \tag{3.3}$$

where N is the number of samples distributed over the hemisphere, ω_i is a sampled direction and $p(\omega_i)$ the probability the direction be chosen. Figure 3.3 illustrates typical distributed ray paths; the grey areas represent the distribution of rays with origin at the apices of the respective triangles. In both examples (paths a and b) the end point of the shadow rays is distributed on the surface of the area light. The paths generated by distributed ray tracing are expressed by $LD?(S|G)^*E$. The algorithm traces multiple rays per intersection, resulting in a combinatorial explosion in the number of rays as the depth of the traversal, and consequently the recursion, increases. A number of terminating conditions such as maximum depth or Russian roulette (Dutre *et al.*, 2003) can be introduced in order to limit the size of the recursion tree.

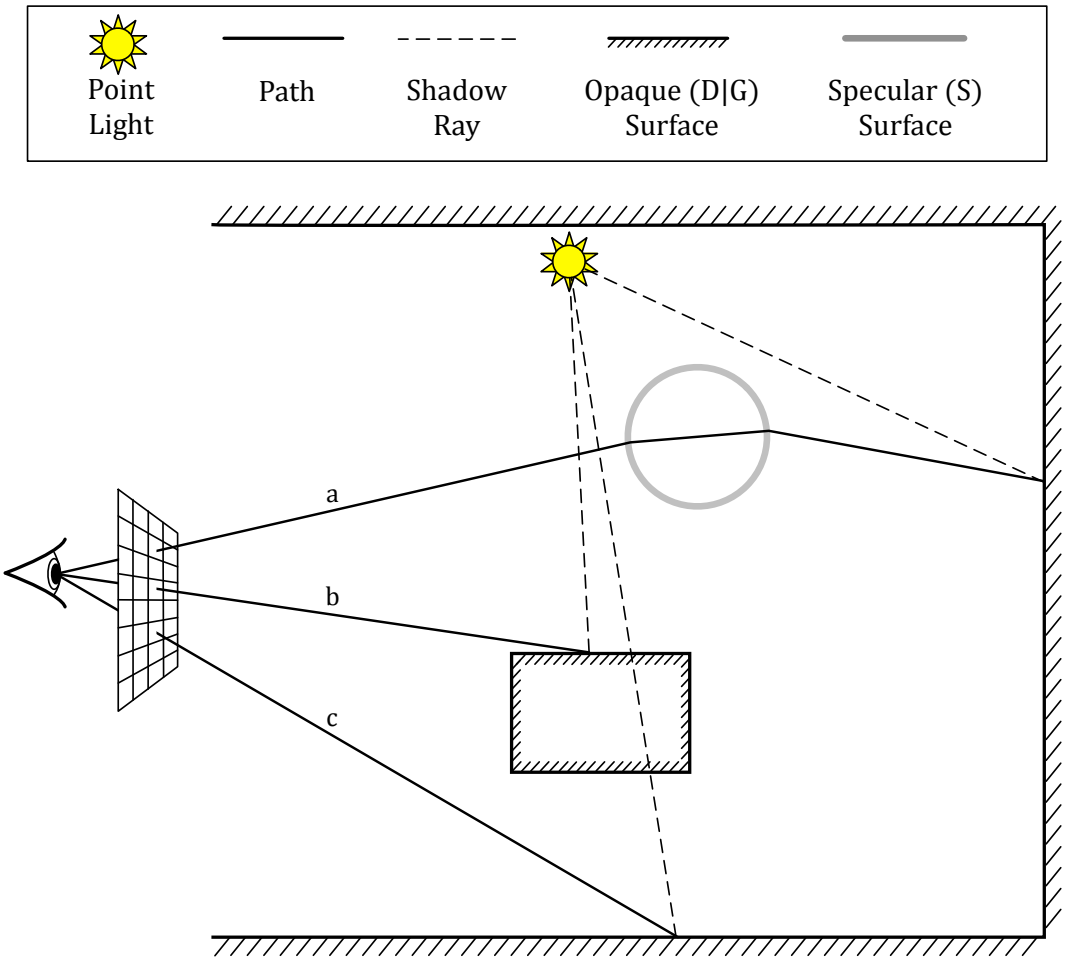


Figure 3.2: Whitted-style ray tracing $[L(D|G)?S^*E]$: path a shows an $LDSSE$ traversal of the scene from light (L) to eye (E); path b illustrates an LDE traversal, while path c never reaches the light, being shadowed.

3.4.2 Path Tracing

Path tracing is a complete solution to the rendering equation (see §2.44), introduced by Kajiya (1986). It allows full global illumination effects to be simulated, including all possible interreflections between different types of surfaces. It is a conceptually simple solution where a ray is traced from the eye into the scene and its contribution used to shade the relevant pixel in the image. Whereas in distributed ray tracing, a primary ray forks into a tree of secondary rays, in path tracing, a single path (Markov Chain) is traced through the scene and eventually terminated using criteria similar to those employed in distributed ray tracing for curtailing the depth of traversal (see Figure 3.4). Generated paths are expressed

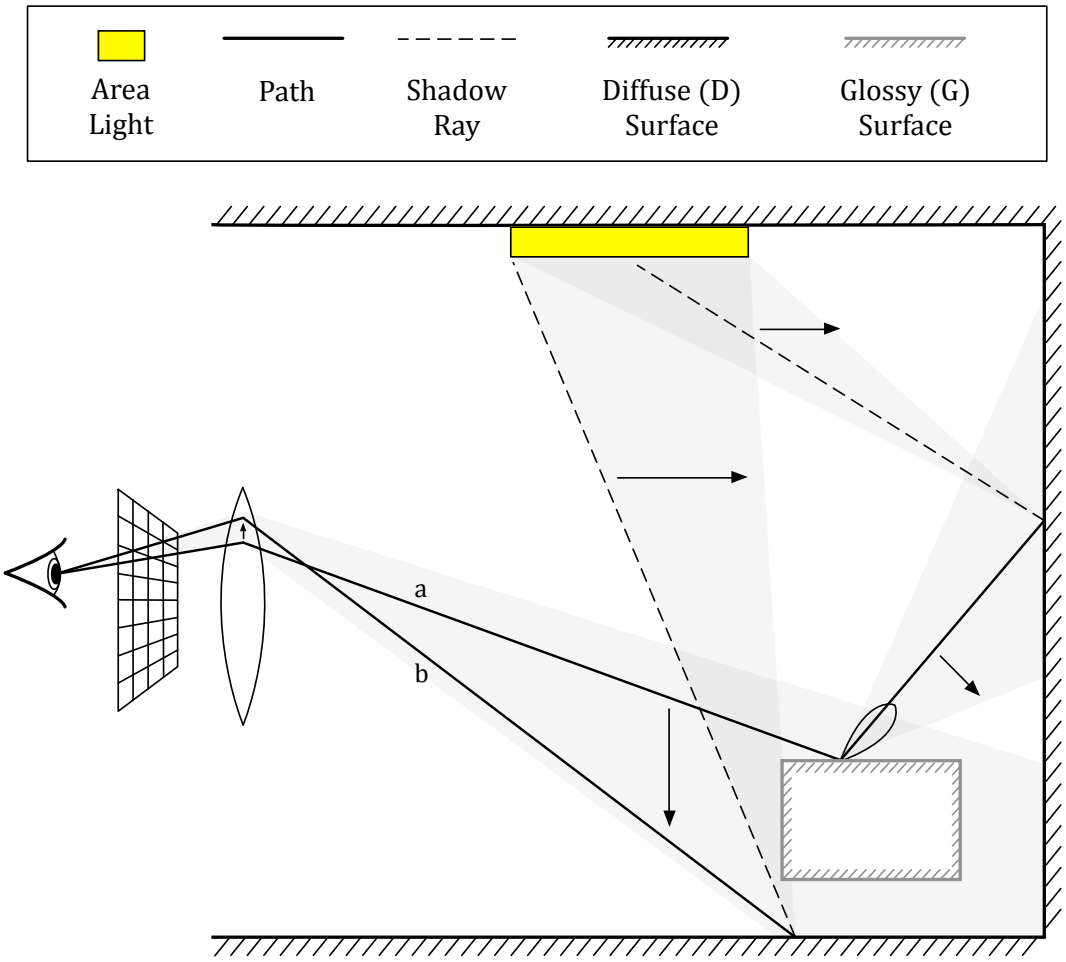
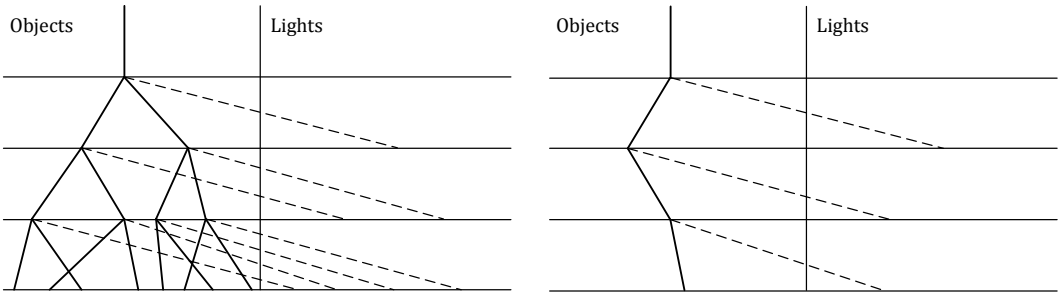


Figure 3.3: Distributed ray tracing $[LD?(S|G)^*E]$: path a shows an $LGDE$ traversal of the scene from light (L) to eye (E); path b illustrates an LDE traversal, with the point on the diffuse surface being partially occluded from the light, thus sitting in the light's penumbra.

by $L(D|G|S)^*E$ (see Figure 3.5).

3.5 Accelerating GI in Stochastic Ray Tracing

Rendering complex environments using stochastic ray tracing strategies such as distributed ray tracing (§3.4.1) and path tracing (§3.4.2) is a very computationally expensive process. Many algorithms have been proposed to enhance the performance of these techniques, such as bidirectional path tracing (Lafortune & Willems, 1993), Metropolis light transport (Veach & Guibas, 1997), photon mapping (Jensen, 2001), irradiance caching (Ward *et al.*, 1988), instant radios-



(a) Distributed Ray tracing (b) Path tracing

Figure 3.4: Scene traversal based on Kajiya (1986). In 3.4a, a single ray branches into multiple, leading to a combinatorial explosion in the number of rays at greater traversal depths. In path tracing (3.4b) a single path is computed, at the expense of higher variance in the solution.

ity (Keller, 1997), instant global illumination (Wald, Kollig, Benthin, Keller & Slusallek, 2002), and instant caching (Debattista *et al.*, 2009); this section provides an overview of the rendering algorithms that are relevant to this work.

3.5.1 Irradiance Cache

Ward *et al.* (1988) observed that diffuse indirect illumination is view independent, and furthermore, irradiance tends to change slowly over a surface when the direct component and its associated shadows have already been accounted for. Thus, they proposed to discretely sample irradiance in such a way as to minimise error and employed a strategy that resulted in more samples being taken at geometric edges. When the irradiance value at a particular point in the scene is not known, it is extrapolated from the set of known irradiance values. The set of irradiance values, or *irradiance samples*, is stored in a data structure called the *irradiance cache*, and is computed on demand. Whenever a new irradiance sample is required, it is computed using an expensive distributed ray tracing pass (see Figure 3.6).

The generation of new points follows an interpolation error metric, based on the geometric properties of the point $\mathbf{x}$ for which irradiance is being extrapolated and the point $\mathbf{y}$ whose irradiance is being interpolated:

$$\epsilon(\mathbf{x}, \mathbf{y}) = \frac{|\mathbf{x} - \mathbf{y}|}{r} + \sqrt{1 - N_x \cdot N_y}, \tag{3.4}$$

where r is the *average* distance to surfaces at $\mathbf{x}$, and N_x and N_y are the sur-

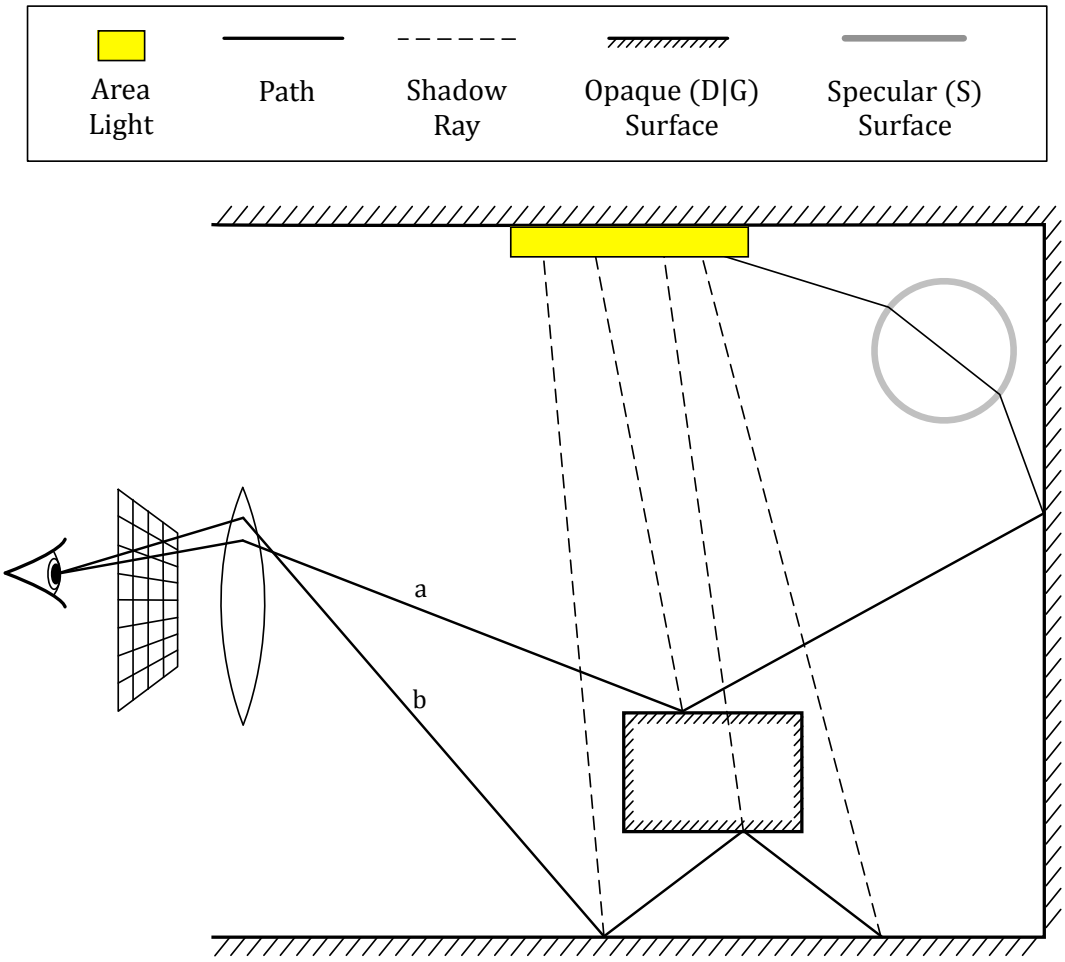


Figure 3.5: Path tracing $[L(D|G|S)*E]$: path a shows an $LSSDDE$ traversal of the scene from light (L) to eye (E); path b illustrates an $LDDDE$ traversal, with some points of intersection falling on the penumbra and another on the umbra of the light.

face normals at positions $\mathbf{x}$ and $\mathbf{y}$ respectively. The error metric ϵ takes into consideration the distance between the two points $\mathbf{x}$ and $\mathbf{y}$, normalised by r. In Ward *et al.* (1988), r is computed by taking the harmonic mean length of rays traced from the cache sample point. The surface normals at the two points are also factored in $(N_x \cdot N_y)$ such that points which are close together in space and have similar normals will be assigned a lower error value. For a point at $\mathbf{x}$, the

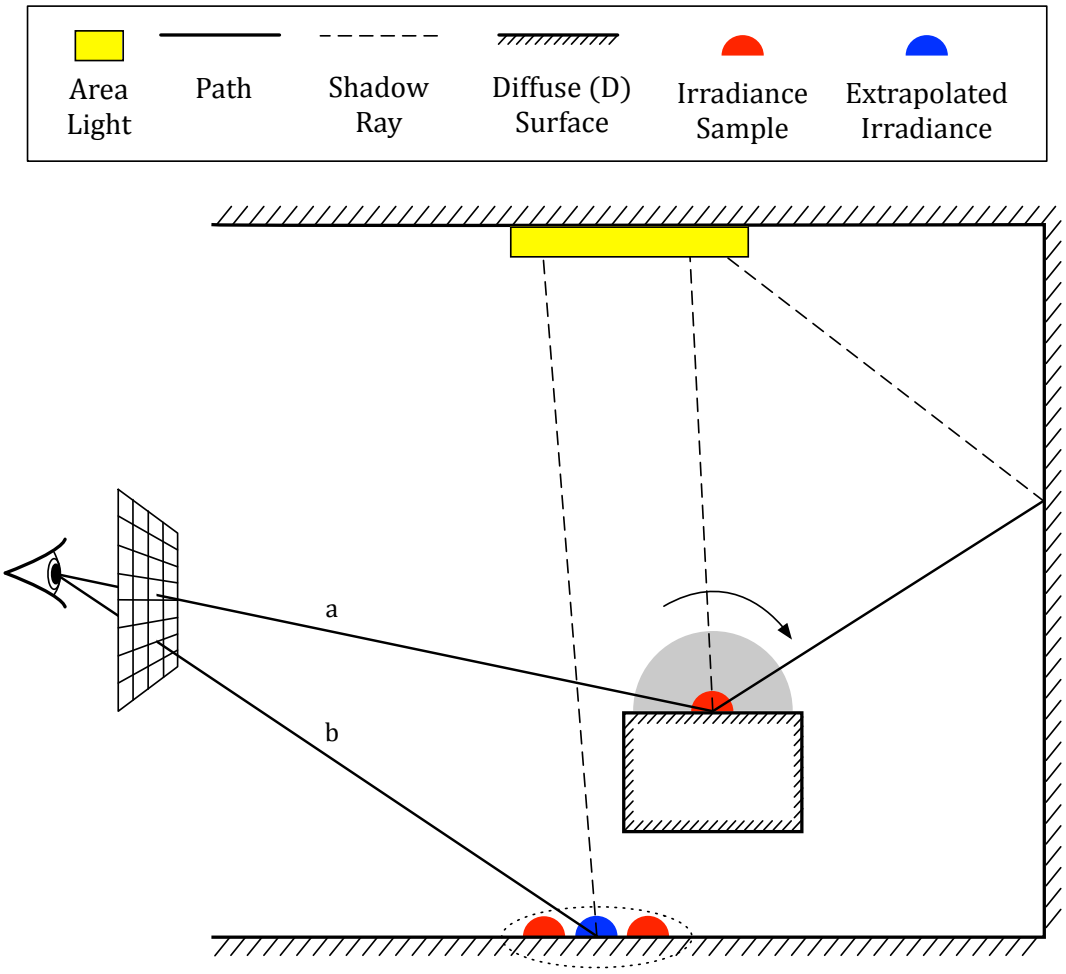


Figure 3.6: Irradiance caching algorithm: path a results in the creation of a new irradiance sample since none of the samples in the cache could be interpolated; path b illustrates the interpolation process whereby two existing samples in the irradiance cache have been used to extrapolate a third (shown in blue).

irradiance $E(\mathbf{x})$ is estimated by:

$$E(\mathbf{x}) = \frac{\sum_{i=1}^{N} \frac{1}{\epsilon(\mathbf{x}, \mathbf{x_i})} E(\mathbf{x_i})}{\sum_{i=1}^{N} \frac{1}{\epsilon(\mathbf{x}, \mathbf{x_i})}} \tag{3.5}$$

where N is the number of irradiance samples considered. The cache lookup for irradiance samples is typically limited to a user-defined radius which defines a sphere centred at $\mathbf{x}$ such that only cache points which are spatially close to $\mathbf{x}$ are considered. Moreover, an octree data structure is used to store the cache points

themselves, taking advantage of their spatial coherence to speed up searches for nearby cache points. Another very important advantage of the irradiance cache is its independence from the underlying geometry.

3.5.2 Instant Radiosity

Instant Radiosity, introduced by Keller (1997), is based on the concept of a particle approximation of the diffuse radiance in a scene. These particles are termed *virtual point lights* (VPL), and are generated using a quasi-random walk based on Monte Carlo integration. The algorithm took advantage of graphics hardware and rendered an image with shadows using each particle as point light source. The results were then summed up using an accumulation buffer (Haeberli & Akeley, 1990). The paths created by instant radiosity are denoted below using α, and are connected to the eye by:

$$P(\mathbf{x}_n) = \alpha \; f_r(\mathbf{x}_3 \to \mathbf{x}_2 \to \mathbf{x}_1) \; G(\mathbf{x}_2, \mathbf{x}_1) \; f_r(\mathbf{x}_2 \to \mathbf{x}_1 \to \mathbf{x}_0) \tag{3.6}$$

where $\ldots \to \mathbf{x}_1 \to \mathbf{x}_0$ denotes a path terminating at the eye (conversely, $\mathbf{x}_0 \to \mathbf{x}_1 \to \ldots$ denotes a path starting at the eye), and α is given by:

$$\begin{aligned} \alpha = & \frac{L_e(\mathbf{x}_n \to \mathbf{x}_{n-1}) f_r(\mathbf{x}_n \to \mathbf{x}_{n-1} \to \mathbf{x}_{n-2}) \; |\cos(\theta_{n-1})|}{p_A(\mathbf{x}_n)} \\ & \times \left(\prod_{i=3}^{n-2} \frac{f_r(\mathbf{x}_{i+1} \to \mathbf{x}_i \to \mathbf{x}_{n+2}) \; |\cos\theta_i|}{p_\omega(\mathbf{x}_{i+1} - \mathbf{x}_i)} \right) \end{aligned} \tag{3.7}$$

In Equation 3.6, the geometry term G may give rise to a weak singularity when the distance between the point being shaded and the VPL decreases. For mutually visible points $\mathbf{x}$ and $\mathbf{y}$, $\lim_{\mathbf{x}\to\mathbf{y}} G(\mathbf{x}, \mathbf{y}) = \infty$. Thus, the closer the two points $\mathbf{x}$ and $\mathbf{y}$ are, the larger the contribution of the geometry term, which can become very large resulting in bright artefacts in the synthesised image. Two solutions have been proposed to mitigate the problem. The first solution proposes to ignore a VPL if it lies within some predetermined range of a point being shaded. The second suggests clamping the geometry term, or the distance in the geometry term, at a minimum value, to avoid the singularity, although this introduces bias into the solution. Kollig & Keller (2006) suggest an alternative method which bounds the radiance integrand and traces a new path to compensate for the introduced bias.

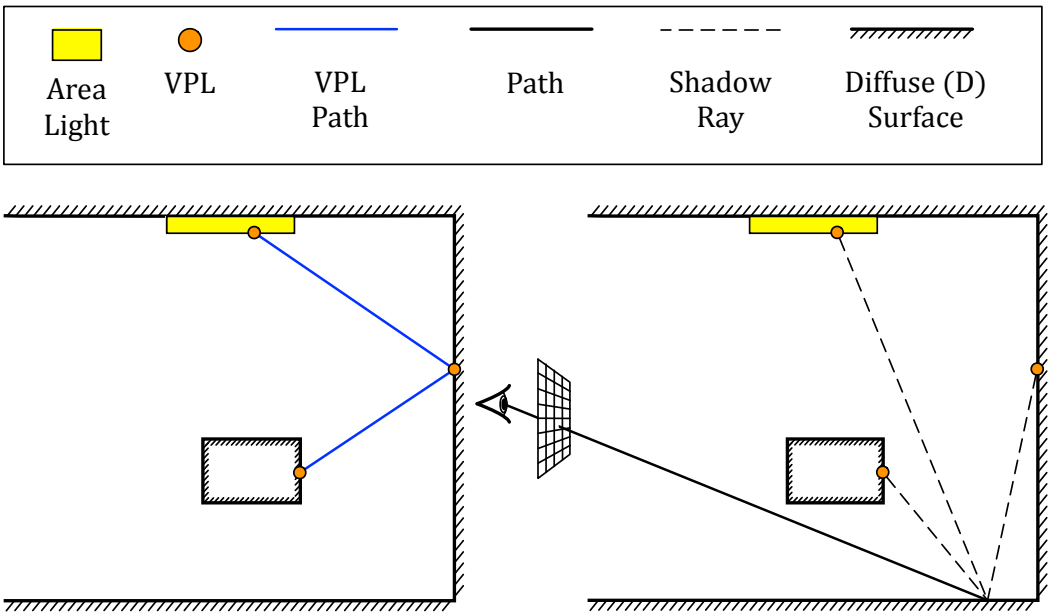
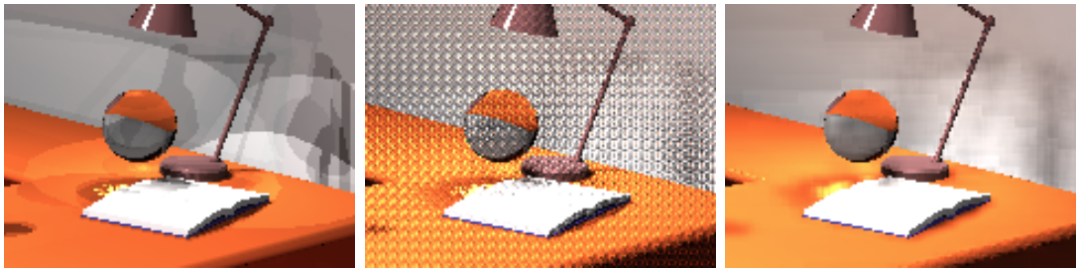


Figure 3.7: Instant radiosity is a two pass algorithm; in the first pass, shown left, a number of virtual point lights (VPL)s are traced and recorded. In a second pass, the VPLs contribute diffuse indirect lighting of their intersected surfaces to the rendering.

3.5.3 Instant Global Illumination

Wald, Kollig, Benthin, Keller & Slusallek (2002) extended instant radiosity and applied it within a real-time ray tracing framework. They proposed a two-pass algorithm, where in a first pass, VPLs were traced from the light source, and in a second pass, for each primary ray intersecting a surface with a diffuse component, they add the contribution of the VPLs. The system was designed for interactivity and during times of no interaction accumulated the images progressively over time to refine the solution and provide anti-aliasing. In order to reduce aliasing artefacts (Figure 3.8a) without generating a different set of VPLs for each pixel, interleaved sampling was introduced. The image plane is subdivided into a set of $m \times n$ pixel tiles, where each pixel is assigned a different set of VPLs, P_k where $1 \leq k \leq mn$. The aliasing artefacts are replaced by structured noise when interleaved sampling is applied, and if the number of VPLs in each set P_k is small, then variance in the resulting images is rather high (Figure 3.8b). Thus, taking into account that irradiance is a piecewise smooth function, Wald, Kollig, Benthin, Keller & Slusallek (2002) use a discontinuity buffer to reduce the noise. For each pixel, the geometric features of the eight neighbouring pixels are



(a) Aliasing artefacts (b) Interleaved sampling (c) Discontinuity buffer

Figure 3.8: Interleaved sampling and the discontinuity buffer are used to reduce aliasing artefacts resulting from the use of a limited number of VPLs for the diffuse indirect contribution. Images taken from Wald, Kollig, Benthin, Keller & Slusallek (2002).

considered; if geometric continuity is detected, the neighbour's irradiance is added to the center pixel's. Geometric continuity between two pixels is determined by taking the dot product of the normal at their centre. The final irradiance for a pixel is computed as the average of the accumulated irradiance multiplied by the reflectance function at that point (Figure 3.8c).

3.5.4 Instant Caching

Debattista *et al.* (2009) extended the irradiance cache (§3.5.1) and instant global illumination (§3.5.3) to propose *instant caching*, an algorithm for interactive global illumination based on instant radiosity methods. The algorithm works in two passes: a first pass traces photons from light sources to generate VPLs. In a subsequent gathering pass, the VPLs are used to compute diffuse indirect lighting at a sparse set of points generated on demand, while the remaining are interpolated, similarly to the irradiance cache. Since VPLs are seldom well distributed over the hemisphere, the harmonic mean distance will not necessarily give an indication of the density of surrounding geometry. Thus, the original error metric formulation of the irradiance cache has been redefined, dropping the mean harmonic distance:

$$\epsilon'(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| + \sqrt{1 - N_x \cdot N_y}, \tag{3.8}$$

where N_x and N_y are the normals at two points $\mathbf{x}$ and $\mathbf{y}$, respectively. The

indirect diffuse component is given by:

$$L_{indirect}(\mathbf{x}, \omega_o) = \sum_{k=1}^{N} f_r(\mathbf{x}, \omega_o, \omega_k)\ L_{e,k}\ V(\mathbf{x}, \mathbf{y}_k)\ G'(\mathbf{x}, \mathbf{y}_k), \tag{3.9}$$

where N is the number of VPLs, V is the visibility function between two points, ω_k is the light vector for the k^{th} VPL, and G' is the bounded geometry term, which is defined as:

$$G'(x, y) = \frac{\cos\theta_x \cos\theta_y}{|\mathbf{x} - \mathbf{y}|^2}\ f_s(0.8 \cdot \min{}_d, 1.2 \cdot \min{}_d, |\mathbf{x} - \mathbf{y}|), \tag{3.10}$$

where θ_x is the angle between ω_k and the normal at $\mathbf{x}$, θ_y is the angle between ω_k and the normal at $\mathbf{y}$, min_d is the bounding distance, and f_s is the smoothing function:

$$f_s(a, b, x) = \begin{cases} 1 & \text{if } x > b \\ 3\left(\dfrac{x-a}{b-a}\right)^2 - 2\left(\dfrac{x-a}{b-a}\right)^3 & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

Equation 3.9 is expensive to evaluate and is used only when computing new samples that go in the cache, Ψ. When interpolating from a set of cached samples in Ψ, Debattista *et al.* (2009) use the following reformulation, which is reminiscent of Ward *et al.* (1988), instead:

$$L_{indirect}(\mathbf{x}, \omega_o) = \frac{\rho}{\pi} \sum_{k \in S(x)} \frac{E_k\ w_k(\mathbf{x})}{\sum_{k \in S(x)}\ E_k\ w_k(\mathbf{x})}, \tag{3.12}$$

where $w_k(\mathbf{x}) = 1/\epsilon'_k(\mathbf{x}, \mathbf{y}_k)$, $S(\mathbf{x}) = \{k \cdot k \in \Psi \wedge w_k(\mathbf{x}) > 1/a\}$, a is the caching radius, and E_k is the cached irradiance for the k^{th} sample in Ψ.

3.6 Finite Element Methods (Radiosity)

The principal finite element method for solving the rendering equation is the *radiosity* algorithm, introduced into computer graphics from thermal engineering by Goral *et al.* (1984). Radiosity is a view-independent method which describes the amount of illumination leaving one surface and reaching another. The radiosity algorithm has been used to accelerate diffuse indirect lighting computations

in both ray tracing (Wallace *et al.*, 1987) and rasterisation-based methods (Baum & Winget, 1990).

Classical radiosity (Goral *et al.*, 1984; Nishita & Nakamae, 1985) only accounted for perfectly diffuse surfaces as it was limited to light paths (originating from light sources) which are reflected zero or more times by perfectly diffuse surfaces before reaching the eye $[LD^*E]$. For perfectly diffuse surfaces, the outgoing radiance at a point $\mathbf{x}$ is uniform across the hemisphere Ω:

$$L_o(\mathbf{x}, \omega_o) = L(\mathbf{x}) \tag{3.13}$$

and thus, differential irradiance $\mathrm{d}E(\mathbf{x}, \omega)$ becomes:

$$\mathrm{d}E(\mathbf{x}, \omega) = L(\mathbf{x}, \omega) \cdot \cos\theta \, \mathrm{d}\omega = L(\mathbf{x}) \cdot \cos\theta \, \mathrm{d}\omega, \tag{3.14}$$

and irradiance $E(\mathbf{x})$:

$$E(\mathbf{x}) = \int_\Omega L(\mathbf{x}) \cdot \cos\theta \ \mathrm{d}\omega \tag{3.15}$$

$$= L(\mathbf{x}) \int_\Omega \cos\theta \ \mathrm{d}\omega \tag{3.16}$$

$$= L(\mathbf{x}) \cdot \pi \tag{3.17}$$

From the area formulation of the rendering equation (Equation 2.45):

$$L(\mathbf{x}) = L_e(\mathbf{x}) + \rho(\mathbf{x}) \int_S L(\mathbf{x}') \ G(\mathbf{x} \to \mathbf{x}') \, \mathrm{d}s \tag{3.18}$$

which gives the radiosity equation:

$$B(\mathbf{x}) = B_e(\mathbf{x}) + \rho(\mathbf{x}) \int_S B(\mathbf{x}') \ G(\mathbf{x} \to \mathbf{x}') \, \mathrm{d}s \tag{3.19}$$

The surfaces can be approximated by a finite number of planar patches with constant radiosity B_i and reflectivity ρ_i:

$$B_i A_i = B_e A_i + \rho_i \sum_{j=1}^{n} B_j F_{ij} A_j \tag{3.20}$$

where F_{ij} is the form-factor from j to i, or the fraction of energy leaving patch j which arrives at patch i, and A_i, A_j are the areas of patches i and j respec-

tively. Due to the reciprocity relationship between form-factors ($F_{ij}A_i = F_{ji}A_j$), dividing by A_i throughout gives:

$$B_i = B_e + \rho_i \sum_{j=1}^{n} B_j F_{ij} \tag{3.21}$$

In order to compute the form-factor for a patch, the visibility between the patch and all patches over the hemisphere of directions above the patch must be determined. This was usually carried out by projecting all other patches onto a hemisphere centred at the patch; Immel *et al.* (1986) proposed the use of a hemi-cube placed over the patch instead, and used standard scan-coversion and hidden surface removal techniques in the projection. Ray-casting methods have also been employed in the evaluation of the form-factors (Sillion & Puech, 1989). The radiosity equation for all patches in the scene may be expressed in matrix-form (Nishita & Nakamae, 1985) and solved as a system of n simultaneous linear equations, where n is the number of patches. To solve the radiosity matrix equation, iterative methods can then be used, such as Gauss-Seidel relaxation or Jacobi iteration.

Immel *et al.* (1986) extended the classical radiosity algorithm to include non-diffuse surfaces. Cohen *et al.* (1988) modified the algorithm to compute form-factors on-the-fly by basing their approach on rendering by progressive refinement. Sillion & Puech (1989) presented a two-pass method for radiosity which integrated specular and diffuse reflection, introducing what they term *extended form-factors* that allowed arbitrary geometries to be used in scene descriptions as well as refraction effects.

3.7 Accelerating GI in Rasterisation

The rasterisation pipeline takes a predominantly object-order approach to image synthesis (§3.3). Efficient indirect lighting algorithms are designed to exploit the nature of the pipeline, mostly by harnessing the power of GPUs and their ability to quickly perform a large number of similar computations in parallel. Yu *et al.* (2009) show that although perceptually important, an accurate computation of indirect illumination is seldom necessary. This philosophy is reflected in the most prominent indirect lighting algorithms for rasterisation, which sacrifice accuracy, such as the number of bounces of indirect lighting, for speed and efficiency.

3.7.1 Radiosity

Dong *et al.* (2007) achieved near real-time frame rates using a method similar to hierarchical radiosity, where hierarchical links are constructed between scene elements and the mutual visibility between them evaluated. Dachsbacher *et al.* (2007) proposed another technique similar to hierarchical radiosity, but eliminate explicit visibility computations using the concept of anti-radiance. The rendering equation is recast in terms of the propagation of radiance and anti-radiance (a quantity used to account for occluders), and solved similarly to the radiosity problem, as a finite element solution.

3.7.2 Precomputed Radiance Transfer

Precomputed Radiance Transfer (PRT), introduced by Sloan *et al.* (2002), uses a low-order spherical harmonic (SH) basis to represent light interaction and transport in low-frequency lighting environments efficiently without aliasing. PRT is based on the assumption that all surfaces in the scene are reflectors, not emitters, and all lights are infinitely distant, making incoming light direction independent of the position of the point being lit. PRT can account for shadows with penumbras (soft shadows) and light interreflections at interactive rates. The fixed relative positions between objects limit the method's application to static scenes - scenes where the objects are fixed but not the observer - and thus cannot be applied to dynamic scenes. Iwasaki *et al.* (2007) treated the objects as secondary light sources; specifically, the intensity distribution of the secondary light sources was represented as a linear combination of basis functions, termed *basis radiances*. These basis radiance functions were precomputed at sample points around the object, resulting in a *basis radiance field*, which was then used to further calculate a *basis irradiance* at runtime by integrating the precomputed basis radiance functions. Sloan *et al.* (2007) introduced an image-based PRT algorithm for accumulating indirect radiance from spherical proxies in environmental lighting, which resulted in a faster approach than previous object-based ones.

3.7.3 Image Space/Instant Radiosity

Dachsbacher & Stamminger (2005) introduced reflective shadow maps (RSM), a technique for computing a rough approximation for one-bounce indirect-lighting in a scene. For every light source in the scene, an extended shadow map is

created to store reflected light, in addition to depth information. The pixels in the RSM are sampled using a precomputed pattern and used as secondary area light sources to provide indirect illumination to the scene. No occlusion was considered for these secondary sources, but RSM achieved single-bounce indirect lighting at interactive rates. Ritschel *et al.* (2008) proposed a method based on instant radiosity where the visibility tests for each VPL are carried out through the use of a shadow map. Since visibility information does not need to be accurate during the computation of indirect lighting, a point representation of the scene is created and used in the generation of shadow maps. Holes which remain as a result of splatting are filled using a push-pull method. McGuire & Luebke (2009) proposed image space photon mapping (ISPM), a hybrid method that runs partially on the GPU where it computes the initial photon tracing bounce and final gathering stages of the algorithm, while processing intermediate bounces on the CPU. The method achieved interactive frame rates for complex environments. Wang *et al.* (2009) proposed another algorithm based on photon mapping where photons were clustered in order to render scenes interactively. Kaplanyan & Dachsbacher (2010) introduced cascaded light propagation volumes where they use a grid to propagate indirect lighting at interactive rates. The technique limits the visibility computations of secondary lights to geometry directly visible.

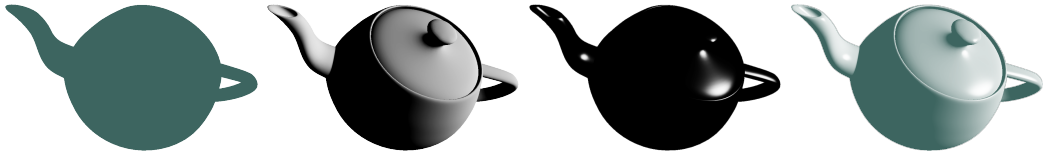
3.8 Empirical Approximations

This section discusses methods that are often used in rendering in place of a physically-based indirect lighting solution. These techniques are empirical in nature and were devised to look plausible rather than correct.

3.8.1 Ambient Lighting

Ambient lighting is one of the three components of the traditional Phong (1975) reflectance model used in real-time rendering, together with specular and diffuse reflection (see Figure 3.9). The ambient contribution is meant to capture slowly varying ambient lighting in the scene, and as a consequence, it is constant throughout the environment, modelled by a single colour value, i_a. The illumination for a surface point $\mathbf{x}$ using these empirical methods is simplified:

$$I(\omega_o) = \rho_a(\mathbf{x})\ i_a + k_e(\mathbf{x}) + \sum_{l \in L} k_d(\mathbf{x}, l) + k_s(\mathbf{x}, l, \omega_o), \tag{3.22}$$



(a) Ambient (b) Diffuse (c) Specular (d) Combined

Figure 3.9: Empirical shading for real-time rendering using Phong (1975). The ambient, diffuse and specular components are combined to get the final result. The ambient component simulates illumination due to diffuse interreflections, the diffuse component direct lighting on diffuse surfaces, while the specular component simulates direct specular reflections from an infinitely far away luminaire.

where k_e gives the emissive contribution, k_d and k_s yield the diffuse and specular light reflected from light source l at $\mathbf{x}$, while ρ_a gives the albedo of the material; $\rho_a(\mathbf{x})\ i_a$ models secondary bounces of illumination.

3.8.2 Ambient Occlusion

The ambient lighting function described above results in flat looking objects where the perception of depth and shape is lost. Ambient occlusion (AO) proposed by Langer & Bülthoff (1999) was introduced to help perceive shape from shading under diffuse lighting. Specifically, the AO function computes how occluded a given point is from ambient lighting. This is given by:

$$A_o(\mathbf{x}) = \frac{1}{\pi} \int_{\Omega} V(\mathbf{x}, \omega)\ \omega \cdot N_x\ \mathrm{d}\omega, \tag{3.23}$$

where V is the visibility function, and N_x is the normal at point $\mathbf{x}$ for which occlusion is to be computed. AO is a global function, and when computed on geometry elements in the scene, the ray-casting function must be evaluated to test whether a given ray $\mathbf{x} \rightarrow \mathbf{x} + t\omega$ contributes to the total occlusion or not, t being the length of the ray segment. The value for t is dependent on the scene for which ambient occlusion is being computed. An example of applied ambient occlusion is given in Figure 3.10. Figure 3.10a and 3.10b show the diffuse and occluded ambient component channels. Areas of the model in Figure 3.10b that are highly occluded appear darker since they receive little ambient lighting. Figure 3.10c shows the ambient occlusion modulating the diffuse shaded model. For an early survey of ambient occlusion techniques please refer to Knecht (2007).

(a) Diffuse (b) Ambient Occlusion (c) Combined

Figure 3.10: Diffuse shading modulated with the ambient occlusion function, giving perceptual clues of depth and spatial proximity.

3.8.3 Screen Space Ambient Occlusion

Mittring (2007) proposed screen space ambient occlusion (SSAO), a faster but less accurate algorithm for computing AO. SSAO provides a number of advantages over a naïve implementation of AO, primarily derived from the use of neighbouring pixels to compute occlusion instead of scene geometry. The SSAO algorithm does not trace visibility rays over the hemisphere of the point for which occlusion is to be computed; instead it analyses the depth values of neighbouring pixels, using differences in depth between the neighbours and the point as a function of occlusion. SSAO is a method that can run entirely on the GPU, without any CPU intervention. It uses point sampling on the depth buffer to determine discrepancies in depth, making it agnostic of the complexity of the scene; sampling the depth buffer is a constant time operation, independent of the number of triangles the scene is composed of. The penalty for SSAO performance comes as a compromise in quality, since the depth buffer is but a numerically limited approximation of the geometry layout in the scene. Concurrently to Mittring (2007), Luft *et al.* (2006) presented an image processing algorithm to provide depth-darkening which resulted in an effect similar to SSAO albeit computationally cheaper. SSAO gave rise to numerous other algorithms for screen space ambient occlusion such as horizon-based ambient occlusion (HBAO) by Bavoil *et al.* (2008), and screen space directional occlusion (SSDO) by Ritschel *et al.* (2009).

3.9 Summary

This chapter has provided an overview of high-fidelity rendering for ray tracing and the rasterisation pipeline, with a focus on the techniques that are relevant to this thesis. A select number of methods for accelerating the computation of GI in stochastic ray tracing were also given, followed by fast but approximative methods for computing indirect lighting using graphics hardware. The chapter concludes with a number of ad hoc, non physically-based methods for approximating indirect lighting, which enhance the depth, curvature and spatial proximity perception of objects within the environment without the associated costs of a full GI solution.

CHAPTER 4

Parallel and Distributed Rendering

This chapter focuses on the use of parallel and distributed systems for ray tracing and global illumination, with a focus on interactivity. It begins by defining speed-up and the main factors constraining it in parallel and distributed systems. Subsequently, problem decomposition for ray tracing methods is examined, and an overview of the master-worker paradigm provided, for both demand-driven and data-driven strategies. Data management in distributed systems is also discussed, leading to considerations of virtual shared memory systems. A review of parallel and distributed rendering literature is then provided, followed by a summary of the chapter.

4.1 Overview

In distributed rendering algorithms, similarly to other distributed computational problems, the goal is that of subdividing the problem at hand such that concurrent resource usage is maximised and the time required to compute a solution is minimised. Many computational algorithms can be split into two parts, a sequential (T_f) and a parallel component (T_p); the sequential component is limited to executing exclusively on a single processing element, while the parallel component can be farmed out to multiple processors for parallel execution. The total execution time T_t of a program executing on P processing elements is given by:

$$T_t(P) = T_f + \frac{T_p}{P}, \tag{4.1}$$

with the total speed-up T_s on P processors defined as:

$$T_s(P) = \frac{T_t(1)}{T_t(P)}. \tag{4.2}$$

Normalising the sequential and parallel components, such that:

$$\gamma = \frac{T_f}{T_t(1)}, 1 - \gamma = \frac{T_p}{T_t(1)} \tag{4.3}$$

yields Amdahl's Law (Amdahl, 1967) from Equation 4.2:

$$T_s(P) = \frac{1}{\gamma + \frac{1-\gamma}{P}}. \tag{4.4}$$

The importance of the sequential component T_f in the maximum speed-up achievable by an algorithm is shown by taking the limit of Equation 4.4, as $P \to \infty$:

$$\lim_{P\to\infty} \frac{1}{\gamma + \frac{1-\gamma}{P}} = \frac{1}{\gamma}. \tag{4.5}$$

There is an upper bound on the maximum speed-up, independent of the number of processors on which the algorithm runs but directly related to the sequential component of the algorithm. Constraints imposed by synchronisation, communication and access to data, which force the various processing elements to wait on some condition to be satisfied before proceeding, contribute to the sequential component and diminish the ability of an algorithm to scale. Problem size and subdivision granularity might also be contributing factors that limit scalability, which is why a sound problem decomposition strategy is extremely important in a parallel algorithm.

The efficiency of a parallel algorithm is given by:

$$e_P = \frac{T_s(P)}{P}, \tag{4.6}$$

where P is the number of processors. The efficiency e_P is typically a value between zero and one; for ideal speed-up, $e_P = 1$.

4.2 Problem decomposition

The subdivision of a problem amongst many processing elements introduces the notion of the *task*, which is the unit of work assigned to a single processor in the system. The *task granularity* of a problem determines the computational effort associated with a task. During the process of problem decomposition, where task granularity is determined, a number of considerations are made, such as the communication frequency between the different processing elements, bandwidth requirements, and computation and data dependencies amongst others. These considerations must be made in the light of the infrastructure that will run the parallel algorithms. In shared-memory systems, the need for communication of data between processing elements is reduced; nevertheless, control on resources accessible by all processing elements needs to be ensured, to avoid inconsistencies due to unpredictable access such as race conditions. On the other hand, in distributed memory systems, there is a need for communication since the processing elements each have their own private memories which cannot be seen or directly accessed by other processing elements.

Crockett (1997) provides a survey of parallel rendering for rasterisation and ray tracing methods amongst others; Chalmers *et al.* (2002) focus specifically on parallel rendering for ray tracing methods. In general, there are a number of approaches used in parallelising the rendering process, although the most common in interactive scenarios are the *functional* and *data* parallelism approaches. In the functional approach, the rendering process is split into several stages, with each stage mapping to some function, or group of functions, that can be applied to an individual data item. Each stage of the pipeline is mapped to a processing element, establishing a data path between the individual processing elements in true producer-consumer style. This leads to the formation of a sequential pipeline, also known as the *rendering pipeline*, where a processing element forwards each completed work item to the processing element assigned to the next stage, while receiving a new datum to process from the neighbour mapped to the previous stage of the pipeline. The functional approach has two significant limitations: the overall speed of the pipeline is determined by its slowest stage and the available parallelism is limited to the number of stages in the pipeline (Crockett, 1997).

The data-parallel approach transposes the functional approach; instead of processing a single data item stream, data is split into multiple streams and operated upon simultaneously. The advantage of this approach lies in its scalability

first and foremost. The number of processing elements employed in any rendering task can vary depending on the problem size, distilled in factors such as scene complexity, image resolution or desired performance levels (Crockett, 1997). The data-parallel approach is further subdivided into *object* and *image* parallelism. Object parallelism denotes operations which are independently carried out on the geometric primitives that make up a scene. Image parallelism, on the other hand, refers to operations used to compute individual pixel values of a synthesised image.

4.2.1 Task and Data Management for Ray Tracing

Task granularity is defined in terms of the smallest unit of computation possible with respect to the problem domain. In the domain of ray tracing-based rendering, Chalmers *et al.* (2002) define the ray-object intersection operation to be the smallest element of computation, or the *atomic element* giving the finest level of granularity. A task is defined as the tracing of one complete path, from the eye to a light source. Path-level tasks provide the fine granularity required by a scalable algorithm; nevertheless, such a level of granularity might result in an accumulation of sequential elements, such as excessive communication per task or frequent accesses to synchronised data structures amongst others, swamping computation time by the effort required to set up the task itself. Thus, it is common practice to adopt an agglomerative strategy to amortise the sequential elements with respect to computation time without seriously compromising the scalability of the algorithm. Chalmers *et al.* (2002) refer to such an agglomeration as the *task packet*, a collection of one or more path-level tasks to be computed.

Ray-casting for visibility determination operates upon geometric object primitives and thus requires scene information to be readily available at the processing element. *World data models* represent problem sizes that fit in the individual private memories of processing elements in a distributed system, where all required scene information is replicated (Chalmers *et al.*, 2002). In scenarios employing this model, no data management is required. On the other hand, problem sizes that do not fit entirely in memory require special external memory (out-of-core) algorithms for efficiently managing data and mapping parts of it to what memory is available at a processing element. This may entail working with data stored on secondary storage or on remote repositories that have to be accessed via a network interconnection. Such implementations may benefit from a virtual shared

memory (VSM) system, which not only provides all processing elements with a single unified address space, but also allows each one to operate on data sets which are larger than their individual private memories, by presenting a *virtual world model* view of the problem (Li, 1988; Li & Hudak, 1989; Chalmers *et al.*, 2002).

A VSM may be provided at different levels, from application all the way down to hardware. For instance, the memory management unit (MMU) in a non-uniform memory access (NUMA) architecture would transparently determine whether a read or write operation is directed at a local or remote memory address and redirect the request accordingly. At operating system level, VSM is usually implemented using mechanisms similar to paging, where the address space is divided into fixed-size chunks, and any access to a chunk that is not available at the local machine would trigger a page-fault, leading the VSM system to fetch the chunk before restarting the faulting instruction. At compiler level, data item sizes may be arbitrary, with the compiler providing data transport and consistency while trying to maximise locality. Finally, at the application level, VSM is provided via data management middleware, which is responsible for servicing any data requests on behalf of the application.

Fundamentally, a VSM provides a shared memory abstraction to systems that communicate via message passing, interpreting and executing special read and write operations such that the shared memory is made consistent across all participants in the system. The efficiency of a VSM is highly dependent on the level of coupling of the processing elements in the distributed system, the interconnecting infrastructure, and the underlying memory consistency model. Strong models, which attempt to order individual operations implicitly are usually less efficient than weaker models, which provide explicit primitives to enforce synchronisation of local and remote memories, thus explicitly ordering groups of operations instead (Mosberger, 1993; Chai, 2002).

4.2.2 Master-Worker Paradigm

A work distribution model that is often used for parallel computing is the *master-worker* (or replicated-worker) paradigm, which is suited to solving computational problems that can be decomposed into a number of smaller nearly identical independent tasks. In the basic master-worker structure, a single master process divides the problem at hand into a number of smaller tasks and then makes them

available to worker processes. The workers, who spend their time waiting for tasks to compute, request tasks from the master, process them, and respond with the results. The master is then responsible for collecting the results and combining them into a meaningful solution (Andrews, 1991; Freeman *et al.*, 1999).

Traditionally, the master-worker paradigm is task-driven, although data-driven approaches have also been researched (Labidi *et al.*, 2012). An inherent advantage of the master-worker paradigm, provided that the chosen task granularity is not excessively coarse, is load balancing. Each worker process may compute a number of tasks, one after the other; as soon as a task is complete, another is requested from the master's centralised pool. While workers are occupied with large tasks, others might be completing several smaller tasks, naturally distributing work across the workers, based on their availability and the size of the workload. This approach favours the unpredictable workload nature of high-fidelity rendering using ray tracing methods. Furthermore, the class of applications that are suited for master-worker scale naturally, such that additional workers can be effortlessly added to a computation, generally speeding it up. The ability to change the number of workers during the course of a computation is a very important property of this paradigm, when taking time constraints into consideration. Computations which consistently fail to meet assigned deadlines can be augmented with more workers. Replication can be used to make the system fault-tolerant by assigning failed tasks to other workers.

A single master process may be unable to handle an increasing number of workers, introducing a bottleneck in the system and adversely affecting its ability to scale. Banino (2006) showed that using multiple masters, arranged hierarchically, can achieve good performance on large-scale platforms where a large number of independent tasks need to be managed. The asymmetry between the master and workers in the paradigm creates a single point of failure at the master, which is undesirable in systems requiring high availability or reliability. Replication and checkpointing techniques can be used to improve fault-tolerance of the system.

4.2.3 Peer-to-Peer Systems

Peer-to-Peer (P2P) architectures have been used in data sharing, collaboration and for information dissemination. The decentralised nature of these systems addresses scalability problems in distributed applications that exist when the

number of clients starts to grow. P2P approaches aimed at sharing resources and information require efficient search mechanisms to locate required information in a timely manner. In local-area solutions, unstructured systems use multicasting facilities provided by the underlying hardware to broadcast queries for specific data. In large scale networks, implementing reliable multicasting is notoriously difficult (Jelasity & van Steen, 2002). An approach adopted by unstructured P2P systems was that of query flooding, whereby all reachable nodes are contacted to determine the availability of a resource on the network. Structured P2P systems such as Chord (Stoica *et al.*, 2001) and Tapestry (Zhao *et al.*, 2001) avoid the traffic caused by query flooding via the adoption of key-based routing and searching. Specifically, a distributed hash table system is used to provide a lookup service similar to an associative array; the search space is partitioned and the search criteria are associated with hosts holding the required resources.

A series of randomised algorithms for replicated database maintenance based on epidemic principles was introduced by Demers *et al.* (1987). This addressed problems of high traffic and database inconsistency, and was later exploited by Demers *et al.* (1994) in Bayou, a system providing support for data sharing and collaboration among weakly connected users, which used peer-to-peer anti-entropy for the propagation of updates. Jelasity & van Steen (2002) conceived the newscast model of computation, providing effective and reliable probabilistic multicasting, large-scale distributed file-sharing, and resource discovery and allocation, with the distinguishing feature being the membership protocol employed. A peer may contact any arbitrarily chosen member and simply copy that member's list of neighbours in order to join a group. Leaving a group is achieved by that peer merely ceasing its communication as opposed to notifying other members in the group about its decision.

4.3 Non-Interactive Ray Tracing

There have been many attempts at parallelising ray tracing and eventually, high fidelity rendering (Crockett, 1997; Chalmers *et al.*, 2002), the first of which were targeted specifically at large supercomputers (Wald *et al.*, 2009). Data-parallel approaches that subdivide problems in image-space, where the entire ray tracing tree of a pixel is processed by a single processing element, are preferred for world data models. Woodwark (1984) introduced a hierarchical method for image-space

subdivision that improved workload distribution across the employed processing elements at the cost of some computation efficiency. Instead of assigning a single contiguous area of a picture to each processing element, a number of pixels on screen are assigned using a recursive subdivision scheme, where each processor would effectively be working at a lower resolution of the actual image, thus achieving more evenly spread workloads between processors. Plunkett & Bailey (1985) proposed a vectorised ray tracing algorithm for a pipelined vector computer which used image-space subdivision to generate a number of ray-object intersection queries. These queries were then queued for processing by the vector processor, achieving very fine-grained parallelism. The initial query set was generated for primary rays; for each ray processed, secondary rays, such as shadow or reflection rays, were generated, queued and eventually processed. The results showed that the vectorised algorithm demonstrated a substantial speed-up, at least an order of magnitude over the scalar implementation. Gaudet *et al.* (1988) proposed a scheme for reducing the complexity of the interconnection network via the use of adaptive broadcasting. Specifically, while arguing both against replicating scene data at each processing element and a global memory, the first due to excessive replication of data and the second due to communication overhead and memory contention resulting from the approach, they proposed to adaptively broadcast data to all processors instead. Their results show that system efficiency declines quickly with an increasing number of processing elements due to a higher latency induced by larger broadcasts.

The object-space partitioning approach entrusts each processing element with the monitoring of a cell or volume in a spatial-partitioning structure. The processing element is responsible for testing all rays that enter the assigned cell against all objects that have surfaces intersecting the cell. Rays that travel from one cell to another assigned to a different processor must be propagated across processors. The propagation of rays across multiple processors, as well as the redundant testing of rays against objects which straddle multiple cells may cause excessive overhead. Another problem of object-space approaches is imbalances in the workload caused by a non-uniform distribution of rays and objects among the processors (Lin & Slater, 1991). Dippe & Swensen (1984) proposed an object-space subdivision algorithm, where the shape of each subregion was adaptively controlled to maintain a roughly uniform distribution of computation load. The subregions were bounded by tetrahedra, forming a general cube, and subject to fixed connectivities. Transfers of load between subregions occurred when the

workload of a region was higher than that of its neighbours and was carried out by moving the vertices of the region's bounding volume; for simplicity, only a corner at a time was moved. The computational effort, or difficulty, involved in shifting the load from one subregion to another was taken into consideration when choosing how to carry out the redistribution. Nemoto & Omachi (1986) argued that load transfer among subregions by moving corners of a general cube affects eight subregions sharing the vertex, making the problem of selecting a corner and choosing a direction and magnitude of movement a difficult operation. Moreover, boundary-intersection calculations for general cubes, as well as the determination of which objects in a subregion should be moved during redistribution are also expensive operations and pose a significant overhead. Thus, they proposed a space subdivision algorithm where subregions are orthogonal parallelepipeds consisting of unit cubes aligned to the coordinate axes of the containing space. Each processing element was assigned to one subregion, thus communicating with six neighbours. Redistribution occurred by moving, or sliding, the boundary surface between two subregions by one unit, transferring the load from the shrinking subregion to the growing.

Salmon & Goldsmith (1989) proposed a hierarchical subdivision of space using rectangular extents; the upper levels of the resulting tree (termed *forest* by the authors) were replicated at each processing element, while the lower levels pointed to the subtrees making up the remaining part of the hierarchy and associated object database, stored at different processors. Each processor also controlled a subset of pixels. A primary ray was initially traced through a pixel and the forest at the originating processor. If the traversal lead to a subtree located at a different processor, the ray was forwarded to the concerned unit; otherwise it was computed locally. Scherson & Caspary (1988) augmented the work of Salmon & Goldsmith (1989) with dynamic load balancing. The traversal and ray-object intersection calculations were decoupled from the other bounding-volume calculations which could run on any processor due to the ubiquitously available forest, to be handled by two different processes. The load balancing technique adopted was that of shifting the traversal of the forest to idle processors when a specific processor was busy computing ray-object intersections. Although Scherson & Caspary (1988) solved the problem of load balancing, their solution was still subject to network congestion caused by the large number of messages exchanged. Priol & Bouatouch (1989) underscored the degradation in performance experienced by previous distributed algorithms due to an increase

in boundary ray intersections and message traffic as the number of processing elements increased. They also note that the large number of messages may cast some processors in a situation of deadlock. Thus, they proposed a static load balancing strategy using image sub-sampling, which is carried out prior to the synthesis phase. The ray tracing of the sub-sampled image acts as a guidance in the subdivision of the scene by means of 3D space partitioning. To avoid the congestion of the communications network, messages representing light rays transitioning from one processor to another are aggregated and replaced by a light volumes in the form of a pyramid. Notwithstanding, the results show that the efficiency of the algorithm rapidly decreases to 30% when the number of processors is increased to 64. This stems from the increase in the number of ray-object intersections; subdivided regions sharing the same objects perform repeated intersection calculations when rays move from region to region. Pitot (1993) proposed another static load distribution strategy where the scene is spatially partitioned into a number of small regular cells, independent of the number of processors. The 3D grid formed from partitioning the scene is then mapped onto a 3D torus, with multiple scattered cells possibly mapping to a single processing element. The subdivision process divides space at two levels; the first, metavoxels, is distributed among the processors. The metavoxels are then divided into voxels that are not distributed. The algorithm was shown to be efficient when synthesising images with complex ray-trees but still suffered from a high communication cost and the rigidity of regular subdivision.

A number of hybrid approaches based on a combination of various degrees of image and object-space partitioning were also put forward. Green & Paddon (1990) compiled a minimum set of design criteria for the development of a flexible and efficient general-purpose multiprocessor solution for ray tracing. They observed that systems exploiting coherence in object-space were either designed specifically for a particular architecture, or as in the majority of cases, the architectures were designed to complement the algorithms used. They proposed a hybrid image-space approach where task granularity is dependent on the size of the image regions used, is demand-driven and thus, automatically load balanced. A local cache employing a direct mapping scheme is held at each processor and keeps a partial view of the object database. The cache is divided into two sets, a statically allocated (*resident set*), and one based on a dynamic mechanism. The resident set holds the objects that were referenced most during the generation of an image. The estimate is computed via the generation of a low-resolution image,

similarly to Salmon & Goldsmith (1989). The results show that an important factor affecting the efficiency of the system was the ratio of dynamic to static storage, and that a larger dynamic section is required as the memory size was decreased. Badouel & Priol (1992) use a dynamic demand-driven image-space partitioning algorithm; the image is initially partitioned into a number of regions equal to the number of processors. When a processor completes its work, it sends a request for more work to another node which is currently busy. Experimental results showed that a 3×3 pixel work item yielded a good balance of communication activity and computation. Furthermore, Badouel & Priol (1992) employed an object-based VSM system (see §4.2.1), to provide their system with the ability to synthesise databases that are larger than the available physical memory. Reisman *et al.* (2000) presented a scheme for ray tracing images at a fixed frame rate by using progressive rendering on distributed systems.

Notwithstanding the partitioning approaches employed, in their survey of load balancing strategies for parallel ray tracing, Heirich & Arvo (1998) conclude that static load balancing strategies result in unacceptably high load imbalances, and thus are non-optimal for use in time-constrained parallel ray tracing on a large number of computers.

4.3.1 Irradiance Cache

Strategies for parallelising the irradiance cache have been proposed both for shared memory and distributed systems. Shared memory approaches can benefit from the use of a single cache that is contemporaneously updated by multiple threads or processes in the system. This essentially makes the irradiance cache a shared data structure, and while helping to avoid work duplication on behalf of each processor, access to the cache must be controlled to prevent any simultaneous access by multiple threads from leaving the data structure in an inconsistent state. Straightforward approaches employ the use of lock-based mechanisms and paradigms, such as readers-writers, which provide mutually exclusive access to the cache. Such an access pattern may create lock contention between the processors, reducing the scalability of the solution. Debattista *et al.* (2011) proposed a wait-free version of the parallel irradiance cache for shared memory systems which avoided the traditional locking approach.

In distributed systems, a parallel irradiance cache must strike a balance between cache misses and communication overhead. The standard radiance distri-

bution (Ward, 1994; Larson *et al.*, 1998) uses the Network File System (NFS) to provide shared access to the irradiance cache in a distributed environment. Contention was dependent on the efficiency of the lock manager used. Koholka *et al.* (1999) shared irradiance sample batches between worker processes after every 50 calculated samples using the Message Passing Interface (MPI). Robertson *et al.* (1999) proposed a master-worker model where, for a predetermined batch size, each worker calculates and stores irradiance samples at the master; the worker would gather samples computed by other workers from the master according to some threshold. Debattista *et al.* (2006) used a component-based approach to partition the computation of indirect diffuse from the other rendering, dedicating a set of nodes to its computation.

4.4 Interactive Ray Tracing

Keates & Hubbold (1995) implemented a custom ray-tracer for a 64 processor machine with virtual shared memory, using a regular grid as acceleration structure. The ray tracer was demand-driven and used a two-level hierarchy for screen subdivision, which determined task packets. The system achieved interactive rates of 1-5 Hz on 32 processors by modelling only primary ray intersections (ray-casting), without any secondary rays traced, and with the use of progressive rendering. Muuss (1995) presented an architecture for interactive ray tracing on a 96 processor system, which achieved 0.5-2 Hz at a resolution of 720×486, modelling three spectral bands. Ray-object intersections were accelerated using binary space partitioning (BSP) trees (Fuchs *et al.*, 1980). Parker *et al.* (1998) developed a custom ray-tracer for rendering iso-surfaces on a 128-processor DSM machine. Ray traversal through the data was based on the incremental method described by Amanatides *et al.* (1987) but further optimised through the use of a multi-level spatial hierarchy to accelerate the traversal of empty voxels. In order to improve locality, for a more efficient cache use, and the reach of translation look-aside buffers (TLB), the volume is ordered into *bricks* (or 3D tiles) to index sparse cells efficiently and only recall the required voxels on demand (Cox & Ellsworth, 1997). The resulting system was highly scalable and could achieve interactive rates, between 1-20 Hz, or 1-10 Hz when shadows were enabled. Parker *et al.* (1999) extended previous work into a more general ray tracing system that could render primitives such as spline models, spheres and polygons. In addition

to conventional rendering, the system also supported frameless rendering, where the pixels are updated according to an asynchronous quasi-random pattern, while the observer and screen are still updated synchronously. Run times for the system ranged from 1-20 Hz at resolutions of 512×512 pixels on 60 processors with a task granularity of 32×4 pixel tiles, although interactivity was managed on as little as 8 processors. Wald, Slusallek, Benthin & Wagner (2001) moved interactive ray tracing from the domain of large supercomputers to a cluster of commodity desktop machines via RTRT, a heavily optimised ray tracing system which made use of instruction-level parallelism and exploited ray coherence through packets of rays in an optimised BSP-tree traversal algorithm. The system achieved interactive rates for scenes of up to 8 million triangles, running a cluster of workstations (CoW) of five Pentium III-class desktop machines interconnected via 100-Mbit Ethernet. Wald, Slusallek & Benthin (2001) extend previous work to allow rendering models of up to 50 million triangles. A pre-processing step was added to the ray tracing pipeline, to subdivide the scene into self-contained voxels. These voxels were then stored on a centralised repository. Clients would then request voxels on demand, and were also responsible for managing a local cache of voxels targeted to exploit spatial and temporal coherence between ray packets. Additionally, computations were reordered so as to avoid waits when voxels were required at a client but not available. Reordering allowed a client to compute other rays while waiting for missing data to arrive, thus hiding transfer latency. Finally, the work distribution approach was based on a task queue of image tiles; the mapping mechanism tried to assign tasks to clients that have traced similar rays in previous frames. The system achieved interactive rates of up to 12 Hz on a cluster of seven dual-core workstations before saturating the network link to the model repository. Wald, Kollig, Benthin, Keller & Slusallek (2002) extended their previous work to provide a full global illumination solution, achieving quasi-linear speed-up on 48 processors.

4.5 Large Scale Distributed Ray Tracing

Distributed approaches to high-fidelity rendering include the use of GRID computing, with algorithms adapted to shared resources. Aggarwal *et al.* (2008) presented a two-stage rendering system for computational grids based on the irradiance cache and, in follow up work (Aggarwal *et al.*, 2009), introduced rendering

on desktop grids wherein single images were computed within user-based time constraints. This work was finally extended to interactive high-fidelity rendering (Aggarwal *et al.*, 2012). Other distributed approaches include BURP (Berkeley Ugly Rendering Project) (Patoli *et al.*, 2009a), which is based on the Boinc framework (Anderson, 2004) and makes use of volunteer computing to perform large-scale rendering. Ramos *et al.* (2009) introduced Yafrid-NG, a physically-based renderer which makes use of a P2P architecture to speed up rendering by distributing computation over the internet to a set of heterogeneous machines. These solutions are nonetheless tailored towards offline rendering, as opposed to interactivity that is the focus of this work. Crucially, all rendering approaches mentioned make use of a master-worker paradigm to distribute computation to worker nodes, notwithstanding the degree of decentralisation employed.

4.6 Cloud-based Rendering

Cloud computing is a model for delivering computing as a service rather than a product. It enables ubiquitous, convenient and on-demand access to a shared pool of resources such as networks, servers and storage, provided to computers and other emerging devices as a utility over a network (Mell & Grance, 2011). In cloud computing, the sharing of resources is fundamental in maximising their effectiveness and achieving economies of scale. Thus, resources are not only shared across multiple users but also reprovisioned on demand. Cloud computing has enabled the use of low performance devices for tasks beyond their computational capabilities. Complex tasks are assimilated into cloud services, allowing applications running on these devices to consume them and, through a messaging pattern such as the request-response model, perform computations on the cloud and receive the results in a fraction of the time it would take the local device to compute. The ability of the technology to scale on demand has been exploited by organisations, both large and small, to provide or consume non-interactive rendering services whilst offsetting the huge associated upfront hardware and infrastructural investments (Baharon *et al.*, 2013). Popular cloud rendering services provide users with an interface to upload and queue rendering jobs for execution on a cloud render farm (*renderRocket*, 2014; *Autodesk 360*, 2014).

In the context of interactive rendering, the entertainment industry has started offering cloud-based computing services for gaming (*OnLive*, 2014; *PlayStation*

Now, 2014; Manzano *et al.*, 2012). A thin client connects to a data-centre in the cloud, where the service provider hosts and runs the actual game, and receives its audiovisual output stream. User input, such as directional controls and button presses, are transmitted by the client to the server, fed to the game and in response, the game output is sent back to the client in the form of a compressed video stream. The bulk of the computation is carried out at the provider's data-centre, allowing a wide range of devices to consume the service, making the computational capacity of the client device largely irrelevant. These systems offer a truly platform-independent experience as the games can be played on any machine with an internet connection.

The remote servers will run a single game per machine, depending on how computationally demanding the game is; in the case of older titles, virtualisation is used to allow running multiple instances of the game on the same machine*. Bandwidth requirements for a number of cloud game streaming services are shown in Table 4.1. These systems offer a service that is equivalent to running the games on a desktop machine with a reasonably good graphics card, and do not attempt to advance the fidelity of the rendering beyond what is currently possible on a single machine. Although effective in providing the same experience to a plethora of devices with varying capabilities, this paradigm is highly susceptible to network latency and bandwidth constraints. High definition (HD) and ultra high definition (UHD) streams, especially at higher frame rates, transfer significant amounts of data (see tables 4.1, 4.2), and may exclude some network configurations due to bandwidth limitations or the introduction of undesired latency in programs that require low response times. In these settings, each client connects to an application that performs the rendering in isolation. This one-to-one approach precludes the possibility of rendering algorithms that amortise computation complexity over a number of concurrent clients, as opposed to multi-user environments.

Service	**Resolution**	**Bandwidth**	**Connection**
OnLive†	576*p*	2 Mbps	WiFi, Wired
OnLive	720*p*	5 Mbps	WiFi, Wired
PlayStation Now‡	720*p*§	5 Mbps	Wired

Table 4.1: Bandwidth requirements for cloud game streaming services.

*http://www.joystiq.com/2009/04/02/gdc09-interview-onlive-founder-steve-perlman-continued/

Pajak *et al.* (2011) propose an efficient compression and streaming algorithm for remote rendering that uses augmented video information, such as depth, to aid in the image reconstruction at client-side. Their solution is also robust to information loss and takes into consideration limited bandwidth connections. The focus of these solutions is primarily on the efficient delivery of interactive streaming content, with the aim of keeping latency at a possible minimum.

Service	**Resolution**	**Bandwidth**
Netflix	SD	2.0 Mbps
Netflix	720p	4.0 Mbps
Netflix	HD	5.0 Mbps
Hulu Plus	SD	1.0 Mbps
Hulu Plus	720p	2.0 Mbps
Hulu Plus	HD	3.2 Mbps

Table 4.2: Bandwidth requirements for various video-on-demand services.

Yet, cloud technologies have the potential for providing a large number of resources to dedicate to a single application at any given point in time, to further the fidelity of interactive rendering beyond what is possible on a single powerful desktop machine. Crassin *et al.* (2013) proposed CloudLight, a system for computing a partial solution to the global illumination problem in the cloud. Particularly, CloudLight computes the indirect lighting component in the cloud to augment interactive rendering on client devices for a full global illumination solution. The focus of CloudLight is low network latency and the amortisation of indirect lighting computations over multi-user virtual environments. Three lighting algorithms, namely voxels (Crassin *et al.*, 2011), irradiance maps (Mitchell *et al.*, 2006) and photon tracing (Jensen, 2001; Mara *et al.*, 2013) are mapped to the system and classified in terms of user compute and bandwidth requirements (see Table 4.3). The results show that irradiance maps scored lowest in terms of user compute and bandwidth requirements (1.4 - 1.6 Mbps), with a modified version of the H.264 codec being used to stream the maps to the clients. The voxel-based solution requires between 3 to 15 Mbps of bandwidth

[†]https://support.onlive.com/hc/en-us/articles/201229050-Computer-and-Internet-Requirements-for-PC-Mac-

[‡]https://support.us.playstation.com/app/answers/detail/a_id/5299/~/playstation-now-network-connection-information

[§]http://www.dualshockers.com/2014/06/10/playstation-now-faq-answers-every-question-you-may-have-pricing-and-more-finally-detailed/

per client and is in the mid-range with respect to computational complexity in the reconstruction of indirect lighting at the client. Finally, the photon tracing solution scored high for both classes, requiring both a relatively powerful client machine and substantial bandwidth, with rates ranging from 25 to 43 Mbps. The irradiance maps strategy is cheap in terms of bandwidth and client compute power requirements, however its application is limited to scenes with appropriate UV-parameterisation, which may be problematic to acquire or define for more complex scenes. As a consequence of this limitation, the authors provide results for half the tested scenes. The voxel-based implementation supports 5 clients at 30 Hz, and 25 at 12 Hz, while the photon tracing implementation shows good scaling for up to 30 clients before the network is saturated. The system was shown to scale up to 50 simultaneous users for a single virtual environment.

Costs	**Voxels**	**Irradiance Maps**	**Photons**
User compute	medium	low	high
Bandwidth	high	low	medium

Table 4.3: CloudLight classifications of indirect lighting algorithms with respect to bandwidth and compute power required for client reconstruction (Low is better).

4.7 Feature/Performance Comparison

Table 4.5 shows a feature and performance comparison for the parallel and distributed rendering algorithms listed in Table 4.4. This table will illustrate where related work does not satisfy certain distributed system characteristics and will help motivate the choice of the methods presented in the next chapters and their design. The comparison is based on a number of common properties of distributed systems:

Interactive The rate at which the rendering system generates new frames. We consider frame rates greater than 5 fps to be interactive (Akenine-Moller *et al.*, 2008), but discriminate between low interactivity ($\leq$ 15 Hz) and high interactivity ($>$ 30 Hz). In the case of streaming systems, the rate of frame generation may be decoupled from that of the video stream and thus, the former value is considered.

Scalable The ability of a system to accommodate growth. Two types of scalability are considered: *horizontal scaling* and *vertical scaling*. Horizontal scaling measures the capability of a system to grow as a distributed system, scaling out to a larger number of nodes, while vertical scaling assesses growth with respect to multithreading, scaling up in terms of usage of the resources of a single node (e.g. number of processor cores). A scalable system can be more easily maintained, and retain reasonable performance when the input or the workload grows.

Elastic The ability of a system to adapt to workload changes by provisioning the available resources such that they match the demand at a specific point in time. Dustdar *et al.* (2011) base elastic processes on explicitly modelling resources, quality and cost; only the first two properties are considered here. A rapidly elastic system provides a better usage of resources as it can react quickly to meet the demands of a varying workload.

M:N A client-server system where the server has the ability to scale out. Horizontal scaling harnesses additional resources when the system is particularly loaded.

Decentralised In a decentralised environment there is no controlling authority and all members perform the same functions, hence they are considered peers. The degree of decentralisation is determined by how strict the demarcation of the roles of each participant in the distributed system are when it comes to providing a service or some management function. A client-server system is considered centralised, while an unstructured peer-to-peer system is considered fully decentralised. *Partial* decentralisation is anything in between. In a decentralised system, as opposed to client-server, peers pool their resources, communicate and collaborate with each other. The decentralisation of functionality can replace powerful central devices in favour of multiple specialised but less powerful ones, while also removing the single point of failure.

Cooperative The cooperation property is defined as the ability of more capable devices to help weaker devices via collaboration. Cooperative systems entail some measure of decentralisation. In a highly cooperative system, weak devices are helped by stronger devices, enabling them to perform computations that would otherwise be unfeasible.

Speed-up Speed-up is the ability of a system to take advantage of multiple processing elements to perform computations in less time (see §4.1). A distinction is made between traditional speed-up and *amortised* speed-up of the method; the former is the performance improvement resulting from parallel computation, while the latter is due to the elimination of redundant computation from appropriate distribution of algorithms.

Bandwidth The bandwidth requirements for a system that performs remote rendering and streams output to the client. This property does not include distributed systems where the master process is also the display process. The classifications of data bands are based on a normalisation of the bandwidths recorded in the systems examined. In particular, *low* $\leq$ 5 Mbps, *high* $\geq$ 25 Mbps and *medium* is the interval in between. The bandwidth requirements of a system impact directly on its performance and running costs. Systems with steep bandwidth requirements scale less favourably.

Precomputation This property denotes whether a specific rendering algorithm requires precomputation before rendering can take place. A system with automatic or no precomputation does not require the user to take any action, while manual precomputation requires user input.

Resolution The maximum recorded output resolution from a given rendering system. The higher the image resolution, the higher the image definition, storage and bandwidth requirements.

It should be noted that Table 4.4 and 4.5 list methods that pioneered interactive ray tracing, against which a performance comparison would be both invalid and unfair. These methods have been included for feature comparison rather than performance, and for the sake of completeness.

4.8 Discussion

Parallel and distributed rendering algorithms (see §4.3, §4.4), with a few exceptions (Aggarwal *et al.*, 2009; Patoli *et al.*, 2009a; Ramos *et al.*, 2009), are designed to assume a priori knowledge of the available resources. Research in rendering on desktop and computational grids has explored the use of non-dedicated resources. Patoli *et al.* (2009b) carried out a preliminary study on creating a render farm

using desktop grids, while Gooding *et al.* (2006) provided an implementation of a distributed rendering system for computational grids; in both cases, the focus was accelerating offline (non-interactive) rendering. Rangel-Kuoppa *et al.* (2003) propose a distributed rendering system for rasterisation, based on a multi-agent platform. They divide the rendering tasks by associating objects with rendering entities. This is reminiscent of object-space partitioning, discussed in §4.3, albeit without the requirement to pass information such as rays for intersection testing, which has been shown to seriously limit scalability as the number of processing elements increased. Gonzalez-Morcillo *et al.* (2010) build on the multi-agent approach to introduce a photorealistic non-interactive rendering architecture that uses image-space partitioning and makes use of importance maps for subdivision guidance and distribution. The focus of their work is not achieving interactivity but efficient load balancing and the optimisation of rendering parameters to achieve shorter rendering times with little compromise on visual quality. Aggarwal *et al.* (2012) propose interactive rendering for desktop grids, dealing with the problem of variable resources also, but only for a single client. Resources shared between multiple clients require addressing the problem of latency incurred when switching between different rendering jobs. This consideration is especially true in the case of rendering jobs using different visualisation algorithms and thus, requiring an altogether different synthesis pipeline be set up for rendering. Green & Paddon (1990) emphasised the existing tight coupling between parallel and distributed rendering algorithms, and hardware architectures, and to a certain extent, this is also present in more recent work such as that by Wald, Kollig, Benthin, Keller & Slusallek (2002), where some generality is sacrificed for efficiency and performance.

The abstraction of shared resources provided by cloud computing has seen new developments arise in the distribution of the parallel rendering pipeline. Game streaming services (*PlayStation Now*, 2014; *OnLive*, 2014) decouple the input and presentation from the rendering, focusing on the reduction of latency due to network performance, instead of improving the quality of the rendering. Pajak *et al.* (2011) proposed a remote rendering service based on a single desktop setup that couples rendering, compression and streaming. In order to aid image reconstruction at the client from low resolution frames and increase robustness to data loss, they augment the streaming of video information using depth and motion information, achieving frame rates of 25-30 Hz at SVGA resolutions (800×600) and 9-12 Hz at HD resolutions (1920×1080) on a weak client (notebook with

a low-end GPU), which makes high-interactivity at HD resolutions impractical on devices such as tablets and smartphones. The system is able to react to network fluctuations and can scale the transmission bandwidth requirements accordingly.

Crassin *et al.* (2013) provide a framework to augment local rendering pipelines with cloud computing processes that provide indirect illumination. Besides focusing on reduced latency in the rendering, they also highlight amortisation of cloud computations for users in the same virtual environment. Three rendering algorithms are provided for computing indirect illumination, each with its advantages and disadvantages (see §4.6). The voxel-based (Crassin *et al.*, 2011) technique is not asynchronous in nature and is highly susceptible to network latency. Irradiance maps (Mitchell *et al.*, 2006) require parameterisation of data sets prior to use, which can prove difficult to acquire for complex scenes (Crassin *et al.*, 2013). The final technique, based on photon tracing (Jensen, 2001; Mara *et al.*, 2013), requires both larger bandwidths (over 25 Mbps) and substantially more client processing power. Besides limits on the type of client that can benefit from this technique imposed by its computational requirements, the larger bandwidth requirements also curtail its scalability.

4.9 Conclusions

Scalability is naturally linked to distributed rendering and to some degree, this is addressed in all previous work. Be as it may, the need for horizontal scaling in more recent work has petered out due to the widespread use of GPUs (*OnLive*, 2014; Crassin *et al.*, 2013; *PlayStation Now*, 2014). In *OnLive* (2014) and *PlayStation Now* (2014), the exclusion of horizontal scaling is by design since neither require rendering capabilities beyond what is achievable on a single machine. Large scale systems such as Patoli *et al.* (2009b), Ramos *et al.* (2009) and Aggarwal *et al.* (2012), which do not utilise GPUs in their synthesis pipeline, scale out reasonably well, and Gonzalez-Morcillo *et al.* (2010) also demonstrate fair scaling for up to eight machines. Patoli *et al.* (2009b), Ramos *et al.* (2009) and Gonzalez-Morcillo *et al.* (2010) do not share the same goals of this work since they target offline non-interactive rendering. Aggarwal *et al.* (2012) support low interactivity for a single client but their system suffers from high initial response times, a side-effect of the grid middleware on which it is based. An idle machine takes minutes to transition to the computation phase once a job has been as-

signed to it (Aggarwal, 2010). Previous work does not address the property of rapid elasticity, which is an essential characteristic of cloud systems.

OnLive (2014) and *PlayStation Now* (2014) provide a low-bandwidth streaming solution that mirrors the requirements of video-on-demand services (see Table 4.2, and strike a balance between better image quality and animation fluidity. Network traffic, stability and latency can drastically affect the input feedback and result in judder and drops in frame rate (Bierton, 2012). With respect to sustained frame rates, *OnLive* (2014) can stream video at 60 Hz, while both *PlayStation Now* (2014) and Crassin *et al.* (2013) are limited to 30 Hz. In terms of bandwidth, CloudLight's irradiance maps are very efficient, but their use is limited by the laborious precomputation step required to create UV parameterisations of scene geometry. There is no solution that provides high-resolution (HD+) streams, high-interactivity ($>$ 30 Hz), low bandwidth ($\leq$ 5 Mbps) and low latency ($<$ 20 ms) ubiquitous delivery of high-fidelity graphics to a vast plethora of device types in previous work.

Decentralised approaches to high-fidelity rendering are only concerned with offline non-interactive systems. Moreover, no proposed system is fully decentralised. Gonzalez-Morcillo *et al.* (2010) use a multi-agent approach to distribute functionality, with the rendering process devolving into multiple groups of master-workers. This is reasonable since the aim of the system is not collaboration but rather, traditional speed-up. Ramos *et al.* (2009) propose a partially decentralised architecture where asset synchronisation between peers was carried out using P2P file sharing, while the actual rendering utilised a master-worker approach. Rangel-Kuoppa *et al.* (2003) use an agent-framework for communication between the different machines, but the system is neither decentralised nor used for high-fidelity rendering. Although partial decentralisation has been enlisted in order to speed-up computation, fully decentralised collaborative rendering is still uncharted territory in previous work.

4.10 Summary

This chapter has provided a literature review of parallel and distributed rendering, spanning offline (non-interactive) and interactive technologies on different infrastructures, from tightly coupled supercomputers to client-server, cloud-based and partially decentralised solutions. The features of these individual systems

Method	**Reference**
Keates & Hubbold (1995)	[1]
Muuss (1995)	[2]
Parker *et al.* (1998)	[3]
Parker *et al.* (1999)	[4]
Wald, Kollig, Benthin, Keller & Slusallek (2002)	[5]
Rangel-Kuoppa *et al.* (2003)	[6]
Patoli *et al.* (2009a)	[7]
Ramos *et al.* (2009)	[8]
Gonzalez-Morcillo *et al.* (2010)	[9]
Pajak *et al.* (2011)	[10]
Aggarwal *et al.* (2012)	[11]
Crassin *et al.* (2013), Irradiance maps	[12*a*]
Crassin *et al.* (2013), Voxel cone tracing	[12*b*]
Crassin *et al.* (2013), Photon tracing	[12*c*]
OnLive (2014)	[13]
PlayStation Now (2014)	[14]

Table 4.4: Work in parallel and distributed rendering.

are identified and grouped synoptically in Table 4.5.

Method	Scalable	Elastic	Interactive	M:N	Decentralised	Cooperative	Speed-up	Bandwidth	Precomputation	Resolution
[1]	✓	—	Low	—	—	—	✓	—	—	—
[2]	✓	—	Low	—	—	—	✓	—	—	486p
[3]	✓	—	Low	—	—	—	✓	—	—	512p
[4]	✓	—	Low	—	—	—	✓	—	—	512p
[5]	✓	—	✓	—	—	—	✓	—	—	480p
[6]	✓	—	✓	—	—	—	✓	—	—	—
[7]	✓	—	—	✓	—	—	✓	—	—	—
[8]	✓	—	—	✓	Partial	—	✓	—	—	—
[9]	✓	—	—	✓	Partial	—	✓	—	—	—
[10]	—	—	✓	—	—	—	✓	Scalable	—	—
[11]	✓	—	✓	—	—	—	✓	—	—	—
[12*a*]	Vertical	—	✓	—	—	—	Amortised	Low	✓	1080p
[12*b*]	Vertical	—	✓	—	—	—	Amortised	Medium	—	1080p
[12*c*]	Vertical	—	✓	—	—	—	Amortised	High	—	1080p
[13]	Vertical	—	High	—	—	—	—	Low	—	720p
[14]	Vertical	—	✓	—	—	—	—	Low	—	720p

Table 4.5: Feature and performance comparison of parallel and distributed rendering systems.

CHAPTER 5

Rendering as a Service (RaaS)

High-fidelity rendering requires a substantial amount of computational power to accurately simulate lighting in virtual environments. It is used in the entertainment industry and for many serious applications such as engineering, architecture, archaeology and defence (Chalmers & Debattista, 2009; Happa *et al.*, 2012). The ability for these fields to be able to compute realistic scenes in real time will enable further productivity to be achieved and potentially open up novel uses of computer graphics. While desktop computing, boosted by modern graphics hardware, has shown promise in delivering realistic rendering at interactive rates, rendering moderately complex scenes may still elude single machine systems. Moreover, with the increasing adoption of mobile devices, which are currently incapable of achieving the same computational performance, there is clearly a need for access to further computational resources that would be able to guarantee a certain level of quality.

Cloud computing is a distributed computing paradigm used for service hosting and delivery. It provides users with a number of advantages, such as the ability to provision a seemingly unlimited number of computing resources without human intervention, location and device independence, and access to the latest software and hardware infrastructures (Mell & Grance, 2011). Cloud systems are responsible for pooling available resources to service multiple users, and through rapid elasticity, allow quick release and reprovisioning of these resources based on user requirements. Cloud computing offers a possibility for users that may not be able to afford to acquire, maintain or manage systems delivering interactive high-fidelity graphics, by providing resources on demand and charging a metered fee based on usage.

Cloud technologies have the potential of providing a large number of resources

to dedicate to a single application at any given point in time. In the context of interactive high-fidelity rendering this could entail dedicating a large number of processing resources to a single rendering task. Existing cloud rendering services do not focus on interactivity, but rather on providing users with compute resources to speed up traditional offline rendering. In order to account for interactivity and to enhance the fidelity of the rendering using distributed computing, a new rendering paradigm is required. This chapter introduces Rendering as a Service (RaaS), a framework that can provide scalable resources for interactive rendering which could be either dedicated or adapt to the servers' workload. RaaS is envisaged as a form of Software as a Service (SaaS) (Mell & Grance, 2011), where users can visualise complex high-fidelity graphics at interactive rates (Akenine-Moller *et al.*, 2008) on relatively underpowered devices via the use of a thin client.

The chapter is structured as follows: §5.1 introduces the chapter and outlines the respective contributions, §5.2 presents a specification for RaaS, §5.3 discusses the resource management aspect of RaaS, §5.4 introduces the concept of the Task Pipeline for distributed work, §5.5 addresses the distributed rendering aspect of RaaS, §5.6 and §5.7 demonstrate results from RaaS followed by a discussion and §5.8 concludes the chapter.

5.1 Introduction

There is a large unfulfilled potential for a system to address the needs of users that require high-fidelity visualisation but lack the resources to do so (Figure 5.1). While a number of cloud services do exist, some used for rendering also, they are not interactive and could not be leveraged to provide interactive rendering as a service, since they do not share the same ambitions. RaaS was architected from scratch with the aim of providing interactivity while adhering to cloud-level properties such as consistency of service. Its viability hinges on the ability to provide, for a given set of resources, various levels of *graphical fidelity*, and *response times* within user expectations. These two properties are dependent; increasing graphical fidelity will invariably increase response time, and lowering response time, sacrifices graphical fidelity. To some degree, parallelism can be exploited to keep response times low while increasing graphical fidelity, but this is largely dependent on the scalability of the rendering algorithms utilised. An added constraint

is that RaaS has to scale well in the number of users also and share computing resources amongst them, as opposed to dedicated rendering systems. Thus, *scalability* and *elasticity* are two important desirable characteristics of RaaS, where scalability is the ability of the system to accommodate growth and elasticity is the system's ability to reprovision a pool of resources in real-time, based on the demands set by the various users. The contributions of this chapter are:

- a specification for scalable and elastic interactive high-fidelity rendering
- a proof of concept for rendering as a service that is flexible, elastic and scalable, for a number of high-fidelity rendering algorithms

5.2 Method

To achieve the goals of rendering as a service, mapping between user clients C and computing resources R should be efficient and flexible. Let P be a set partition of R. A binding $b(c) \mapsto x$ is established between a client $c \in C$ and a block $x \in P$ when at least one computing resource $r \in R$ is assigned to c. The mapping M is the union of all bindings. For example, if two clients c_1 and c_2 connect to an ideal service with computing resources $R = \{r_1, \ldots, r_{12}\}$, and are assigned half the total resources each, then $x_1 = \{r_1, \ldots, r_6\}$, $x_2 = \{r_7, \ldots, r_{12}\}$, $P = \{x_1, x_2\}$ and $M = \{b(c_1) \mapsto x_1,\ b(c_2) \mapsto x_2\}$. Client demands for computing resources fluctuate and RaaS should be elastic enough to respond by transitioning from a mapping M to another M' in real-time. Thus, if a

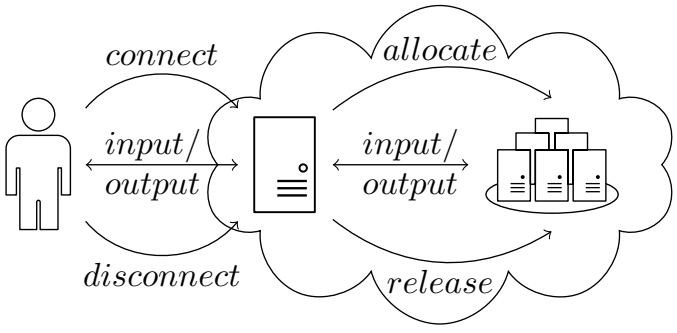


Figure 5.1: Overview of *Rendering as a Service*. Clients connect and have resources allocated to them for the duration of their rendering job, after which, the resources are freed again. All computation occurs remotely, with the server streaming the output.

third client c_3 connects to the system and has equal priority to c_1 and c_2, the system should be able to quickly transition to a partition $P' = \{x_1', x_2', x_3'\}$, where $x_1' = \{r_1, \ldots, r_4\}$, $x_2' = \{r_5, \ldots, r_8\}$ and $x_3' = \{r_9, \ldots, r_{12}\}$, such that the new mapping is $M' = \{b(c_1) \mapsto x_1',\ b(c_2) \mapsto x_2',\ b(c_3) \mapsto x_3'\}$. Such mappings may be performed quickly and efficiently by a central authority privy to both C and R (see Figure 5.2). Client connectivity, and resource management and binding are considered two separate concerns; we propose two entities within the central authority to handle them: the *Service Manager* and *Resource Manager* respectively.

A *job* constitutes the chosen smallest unit of processing that can be managed independently within the framework. It is a collection of computing resources x_m working together towards satisfying the request of a single client c_i, and is described by the binding $b(c_i) \mapsto x_m$. Every job has an associated *Job Controller* (see Figure 5.2), an active entity that relays client requests to bound resources and sends back responses, allowing clients to interact with resources. This is assigned upon client connection. Initially, a connected client has no resources allotted, as a job has yet to be specified. As soon as a client submits a job specification, the system can perform resource allocation (see §5.3). The number of computing resources $|x_m|$ is not directly set by client c_i. Instead, specially designated programs (either automated or user-driven) are used to determine and set resource bindings for each and every client in the system, via a façade exposed by the *Admin Controller* (see Figure 5.2). Administrative and control services are also provided for system management to take place (e.g. load balancing), as well as monitoring tools (e.g. user job monitoring); the service level and functionality exposed is largely determined by the privilege level of the connected user account. Depending on priority and availability, a number of resources may be allocated to the client, allowing for job execution to start. The resources assigned to a Job Controller are all but one given the role of *worker*; the exception is made for the first assigned resource which is given the role of *Task Coordinator* (see §5.4). This arrangement is tailored towards facilitating the integration of parallel rendering algorithms which favour the Master-Worker paradigm (see §4.2.2). It also imposes a hard upper bound c_{max} on the number of clients, for n_{tot} processing elements such that $c_{max} = \lfloor \frac{(n_{tot} - n_{fe})}{2} \rfloor$, where $1 \geq n_{fe} \geq n_{tot}$ is the number of processing elements allocated to the front end. At least a single processing element is required to run the front end service, while each client necessitates a minimum of two processing elements, one running the Task Coordinator and

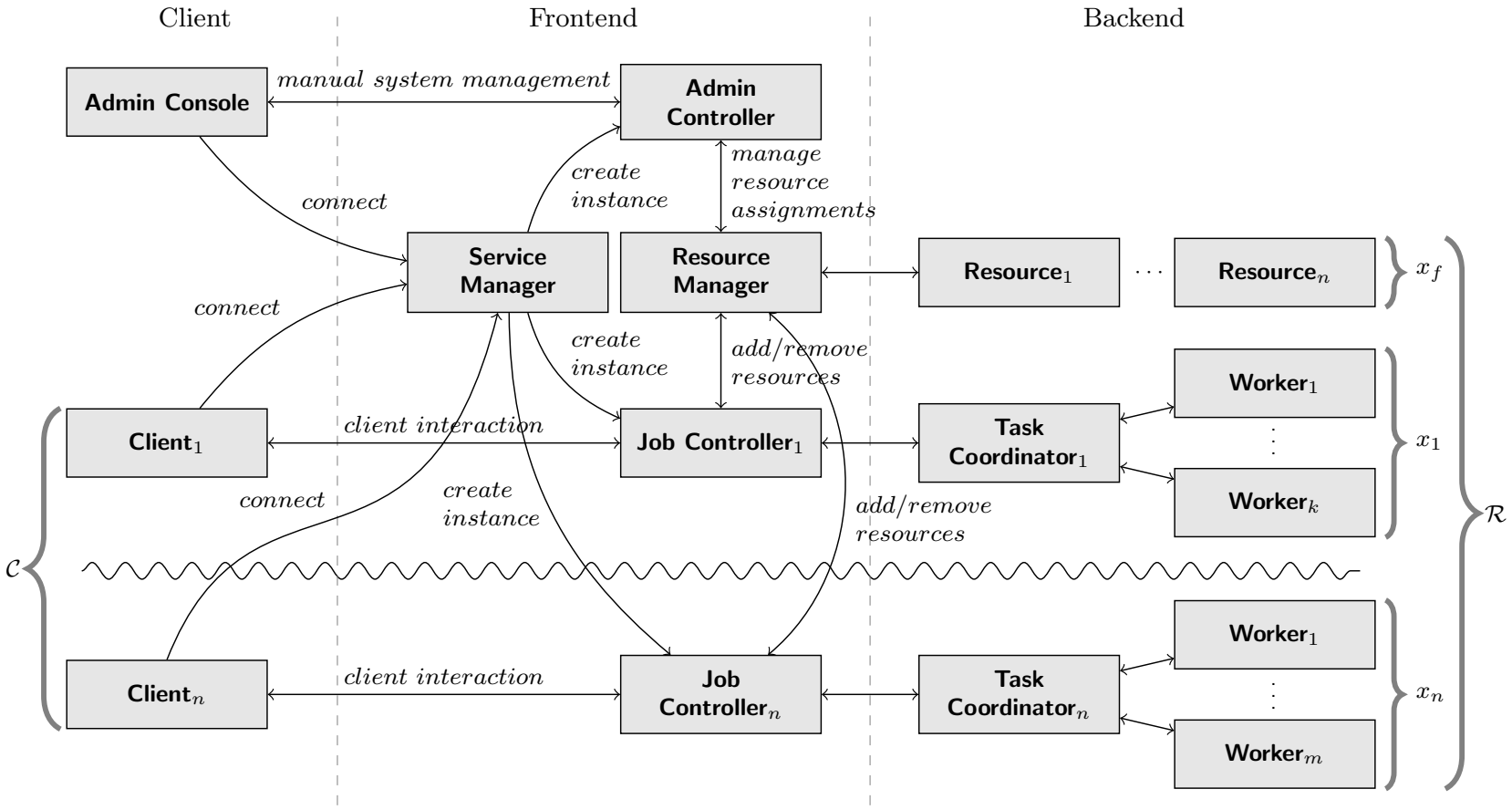


Figure 5.2: High-level architecture diagram for Rendering as a Service that includes the major components of the system. Every connecting client is assigned a Job Controller for mediating communication with the Task Coordinator and Workers, which carry out the actual rendering computation. Note that x_f is equivalent to the free pool.

another a single worker.

5.3 Resource Management

Computing resources managed by the Resource Manager each consist of a single processing element and some measure of private primary memory. A resource may be characterised by two distinct states, *idle* or *busy*. An idle resource can do no useful work unless it is bound to a Job Controller. It resides in an unbound block x_f in P, termed the *free pool*. When a resource is bound (Figure 5.3b), it moves out of the free pool and performs initialisation. For each job, the Task Coordinator is the only resource to receive initialisation instructions from the Job Controller; it must then pass this information on to other workers, to coordinate their initialisation. Moreover it is responsible for disseminating state information (e.g. user input) before each phase of computation, to enforce consistency among all resources belonging to the same Job Controller. If resources are no longer required or required elsewhere, they are unbound from their current Job Controller (Figure 5.3b), after which they shut down, revert to idle state and move back to the free pool.

Resource management is defined in terms of a function L which produces all possible set partitions of R; particularly, let x_s and x_d be blocks in set partition P. The function $L(P, x_s, x_d, c) \mapsto P'$, replaces x_s and x_d by ${x_s}'$ and ${x_d}'$ to give P', such that:

$$T = \{r_i \in x_s | i \in J\} \tag{5.1a}$$

$${x_d}' = x_d \cup T \tag{5.1b}$$

$${x_s}' = x_s - T \tag{5.1c}$$

for some subset $J \subseteq I$ such that $|J| = c$, where I is an index set for x_s and $|x_s| \geq c$. If no partition x_d exists in P, then $x_d = \emptyset$. If ${x_s}' = \emptyset$, then by definition, ${x_s}' \notin P'$. T contains c resources transferred between blocks x_s and x_d to yield blocks x'_s and x'_d. Consequently, x'_s and x'_d become the new x_s and x_d.

5.3.1 Resource Allocation

The aim of *allocation requests* is that of adding resources to Job Controllers, which initially start with an empty pool. Requests are non-blocking; if a request

cannot be satisfied, the call returns immediately. A resource allocation request is considered to be a system management function and as such is initiated through an Admin Controller. The Resource Manager handles the request by attempting to reserve the required number of resources first. Concurrent reservations resolve to a sequential ordering, to protect accesses to shared data structures and prevent interference. After a successful reservation, a message is sent to the Job Controller whose resource pool is to be augmented with the reserved resources. If the pool was empty prior to this event, the Job Controller elects one of the resources as a Task Coordinator. It then broadcasts a message to the assigned resources notifying them they are no longer idle; the broadcast also contains the identification of the elected Task Coordinator (whether elected in this cycle or earlier). On receiving the broadcast, each resource determines its role by checking the message body, and starts executing accordingly. Resource allocation is a specialisation of L, with the added constraint that $x_s = x_f$ (i.e., resources are taken from the free pool):

$$L_a(P, x_d, c) = \begin{cases} L(P, x_f, x_d, c) & |x_f| \geq c \\ P & \text{otherwise} \end{cases} \tag{5.2}$$

5.3.2 Resource Release

Resources are detached from Job Controllers through *release requests*. Like allocation, release requests are initiated through an Admin Controller. When issued, the Resource Manager verifies that the Job Controller in question has enough resources to meet the request; otherwise, the originator is notified immediately and the call ends. The Resource Manager then sends a message to the Job Controller with the size of the resource block it has to relinquish, but leaves it in the hands of the latter to choose the particular resources. The workers are not notified directly by the Job Controller, but by the Task Coordinator, which queues notifications until the brief period of worker quiescence at the beginning of each computation cycle (see §5.4.2). The Resource Manager receives the list of resources from the Job Controller and removes their binding. The resources, once notified, proceed to free allocated memory and disk resources, and perform a shutdown procedure. The Resource Manager does not return resources to the free pool immediately to avoid the loss of allocation messages received by the resources during the shutdown process. When resources complete their shutdown phase, they notify the Resource Manager and switch to idle state. At this point, the Resource Manager

may safely move the resources to the free pool. Job execution will have come to an end if the Task Coordinator itself is included in the list of resources to be freed. Accordingly, having informed all workers, the Task Coordinator performs a shutdown procedure and notifies the Resource Manager that it is idle. Resource release is also a specialisation of L, with the added constraint that $x_d = x_f$ (i.e., resources are added to the free pool):

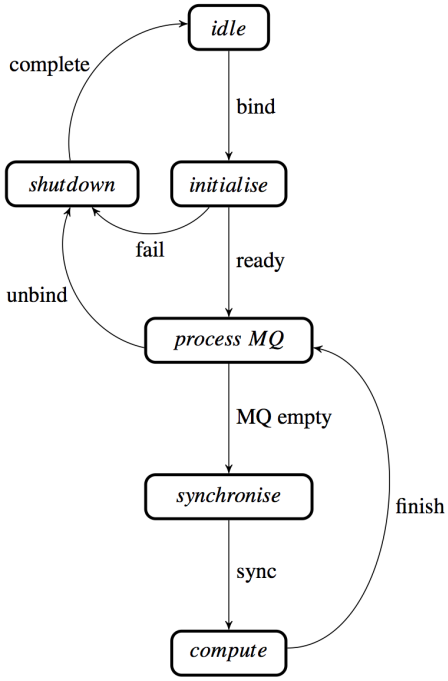
$$L_r(P, x_s, c) = \begin{cases} L(P, x_s, x_f, c) & |x_s| \geq c \\ P & \text{otherwise} \end{cases} \tag{5.3}$$

5.4 The Task Pipeline

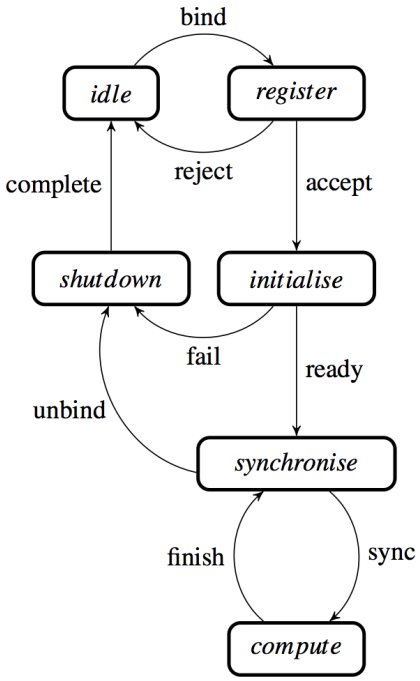
The synchronisation requirements of an active resource that are independent of the rendering process are referred to as *system synchronisation*. System synchronisation is indispensable for ensuring resources are correctly allocated and freed, and that each computation phase is entered simultaneously by the participating resources, amongst others. It is considered fundamental, in that, as a bare minimum, every resource in the system, be it a worker or a Task Coordinator, should be able to handle these protocols for the system to operate correctly. System synchronisation for both workers and Task Coordinators has been abstracted and encapsulated into two pipelines (Figure 5.3a, 5.3b). We refer to the parallel composition of these two pipelines as the *Task Pipeline*.

5.4.1 Initialisation and Registration

Resources that have been elected to become Task Coordinators go through an initialisation stage where their internal state is prepared for the computation at hand. Two mailboxes are set up, the first reserved for messages received from the Job Controller, the second for workers from within the same block. Resources that are assigned a worker role start by registering with the Task Coordinator, where their state is made consistent with the rest of the group. Following a successful initialisation, a worker is in a position to contribute to the rendering computation. Otherwise, it performs clean up and reverts to idle state. Before a computation phase starts, a worker notifies the Task Coordinator that it is ready to accept work; this is referred to as the *synchronisation* stage.



(a) Lifecycle of the Task Coordinator process, showing the transition from idle state to initialisation via binding, the rendering cycle, and shutdown.



(b) Lifecycle of a typical worker process; multiple workers are mapped to a single Task Coordinator.

Figure 5.3: Task Pipeline paradigm illustrating the Task Coordinator and Worker module state diagrams.

5.4.2 Synchronisation

The Task Coordinator and workers are required to start each computation phase together; this is enforced via a timed barrier encapsulated in the *synchronise* state (Figure 5.3) present in both pipelines and explained in detail in Algorithm 4. Initially, the Task Coordinator starts with an empty set of workers, W. A time window is computed, during which workers may signal the Task Coordinator that they can participate in the computation; t_w specifies the size of the window, while t_c represents the current time. While the window is open, messages sent from workers and received in the synchronisation message queue, MQ_{sync}, are processed one by one, and a response r is composed and sent back. For every message m dequeued, the id of the sender is checked against a set of requests to unbind workers, U. Depending on this outcome, the response will contain a flag, r_{unbind}, which tells the worker whether its participation in the next computation phase has been accepted or not; in the latter case, the worker is required to shut-

Algorithm 4 Task coordinator-worker synchronisation procedure to determine resources contributing to frame image synthesis computation.

Require: $min\_workers \geq 0$
Require: t_c increases monotonically
1: $W \Leftarrow \emptyset$
2: $t_e \Leftarrow t_c + t_w$
3: **while** $t_c \leq t_e$ **do**
4: **while** MQ_{sync} is not empty **do**
5: $m \Leftarrow^{deq} MQ_{sync}$
6: **if** $m_{id} \in U$ **then**
7: $r_{unbind} \Leftarrow$ `true`
8: **else**
9: $r_{unbind} \Leftarrow$ `false`
10: $W \Leftarrow W \cup m_{id}$
11: **end if**
12: Send r to m_{id}
13: **end while**
14: **end while**
15: **if** $|W| \leq min\_workers$ **then**
16: **for all** $m_{id} \in W$ **do**
17: Send *abort* to m_{id}
18: **end for**
19: $W \Leftarrow \emptyset$
20: **end if**

down and switch to idle state. If worker m_{id} has been accepted, it is added to W. When the synchronisation period has elapsed ($t_c \geq t_e$), W contains the workers that will participate in the computation. If a lower bound ($min\_workers$) on the number of workers required to start a computation phase has been specified, then it is made sure that $|W|$ satisfies the requirement. If it does not, then an *abort* signal is sent to all workers in W, and W is emptied.

5.4.3 Lazy-loading of Data

Initialisation may require large volumes of data to be loaded into memory and processed before a resource can start accepting work and performing computation. In distributed environments with non-volatile resources, such as dedicated clusters, this cost is offset by having a single initialisation phase at start-up, which covers all resources. Moreover, it is not uncommon for data to be replicated onto faster local storage to reduce initialisation latency even further, as

Algorithm 5 Read operation on scene objects stored on a remote repository.

Require: $b_{size} > 0$ and b_{size} is a power of 2

1: **function** READOBJECT(Object obj, Offset d)
2: $b_{id} \Leftarrow \lfloor d \cdot (log_2(b_{size}))^{-1} \rfloor$
3: $b_{offset} \Leftarrow d \;\&\; (b_{size} - 1)$
4: **if** $b_{id} \notin M_{local}$ **then**
5: $M_{local} \Leftarrow M_{local} \cup b_{id}$
6: $M_{local}(b_{id}) \Leftarrow$ FETCH(obj, b_{id})
7: **end if**
8: **return** $M_{local}(b_{id})[b_{offset}]$
9: **end function**

opposed to having to retrieve data from some remote storage location. Even with such measures in place, it might still take some time before the required data assets are loaded and prepared for processing; for example, it might take seconds to load a highly complex scene and generate the respective structures for accelerating ray tracing. In the case of volatile resources that migrate across different jobs and perform multiple initialisations during their lifetime, a costly initialisation may impinge negatively on the elasticity of the system, which in turn would seriously hinder the ability of a resource to instantaneously join a job and share its workload.

Taking inspiration from page-based distributed shared memory (DSM) systems (Li, 1988; Buck & Keleher, 1998), we adopt a method which reduces data copying and replication, and initialisation latency. In typical page-based DSM, the address space is partitioned into pages, managed by a virtual memory manager (VMM). Whenever a request to a non-local address space is generated, the DSM manager is tasked with fetching the page from the owner, possibly a remote machine. In our case, since computation is done in response to a continuous series of query operations (e.g. ray traversal through acceleration structures), data is organised hierarchically to exploit locality. Specifically, instead of having a single address space containing all the required object data, a strategy similar to paged segmentation is adopted, where each segment represents an object (e.g. geometry and texture data, acceleration structures, etc). Object data structures are generated and persisted in a memory-independent fashion, allowing them to be used directly from secondary storage. Contrary to traditional paged-based DSM systems, we can afford to relax our synchronisation requirements due to a relatively simple and straightforward data access pattern by resources.

When a resource enters its initialisation phase, data objects are mapped into the memory of the resource without being loaded, similarly to pure demand paging virtual memory systems. The first time a resource tries accessing memory within these mapped regions, a data query occurs between it and the repository holding the actual data. A subset of the data, referred to as a *block*, is transferred to the resource, depending on the granularity measure (b_{size}) used for region partitioning, which can then resume computation (Algorithm 5). In this initial proposition, no attempt is made at prefetching or hiding the transfer latency for individual blocks. Once a block has been transferred to a resource, further accesses incur no additional costs whatsoever, unless the respective data object is modified.

5.4.4 Persistence of Temporary Structures

Prior to starting work on a rendering job, a resource often requires to carry out a number of local initialisation actions, such as sorting data to take advantage of some form of coherence present in it. These actions usually result in the creation of data structures which last for the duration of the job. For example, in order to accelerate search operations such as ray intersections with scene geometry, the latter is usually organised into acceleration structures such as kd-trees, to exploit spatial coherence.

In order to reduce start-up latency when resources migrate from the free pool to an assigned job, temporary structures are created at the Task Coordinator and persisted in a repository accessible by all resources pertaining to that job. Shared structures that can be efficiently maintained at each worker resource are not persisted, while objects such as geometry primitives, acceleration structures and texture information are. Persisting texture information is straightforward; uncompressed texture data existing on secondary storage can be easily mapped to memory without any transformation and be immediately usable. On the other hand, acceleration structures have to undergo a more complex transformation in order to be used out-of-core. Algorithm 6 illustrates the method used for persisting a kd-tree containing triangle primitives to secondary storage for out-of-core use in the lazy object loading scheme. A two-pass approach is employed wherein the first pass, the binary tree is traversed and the nodes are persisted to a contiguous buffer. In a second pass, when all nodes have been persisted and their exact offset in the buffer is known, their child node indices are updated

Algorithm 6 Persistence of the out-of-core kd-tree acceleration structure.

Require: $t \neq \emptyset$

1: **function** PERSISTKDTREE(TriangleList t, KDTreeNode $root$)
2: $T_{base} \Leftarrow$ addressOf($t[0]$) ▷ Find base address of first triangle in t
3: $Q \overset{enq}{\Longleftarrow} root$ ▷ Enqueue root node and start first pass traversal
4: $offset \Leftarrow 0$
5: **while** $Q \neq \emptyset$ **do**
6: $node \overset{deq}{\Longleftarrow} Q$ ▷ Dequeue node
7: $M[node_{id}] \overset{map}{\Longleftarrow} offset$ ▷ Map node to current buffer offset
8: $offset \Leftarrow offset +$ size(KDTreeFlatNode) ▷ Increment buffer offset
9: write($node$) ▷ Persist node core
10: **if** IsLeaf($node$) **then** ▷ Persist node elements if leaf type
11: $offset \Leftarrow offset +$ count($node.elements$)
12: **for all** Triangle $t_i \in node.triangleList$ **do**
13: $index \Leftarrow T_{base} -$ addressOf(t_i)
14: write($index$)
15: **end for**
16: **else** ▷ Else enqueue child nodes and persist index placeholders
17: $Q \overset{enq}{\Longleftarrow} node.leftChild$
18: $Q \overset{enq}{\Longleftarrow} node.rightChild$
19: write($indexPlaceholder$)
20: **end if**
21: **end while**
22: $Q \overset{enq}{\Longleftarrow} root$ ▷ Enqueue root node and start second pass traversal
23: **while** $Q \neq \emptyset$ **do**
24: $node \overset{deq}{\Longleftarrow} Q_t$ ▷ Dequeue node
25: **if** isLeaf($node$) $\neq true$ **then** ▷ If not leaf type update node indices
26: $Q \overset{enq}{\Longleftarrow} node.leftChild$
27: $Q \overset{enq}{\Longleftarrow} node.rightChild$
28: seek($M[node_{id}] +$ size(KDTreeFlatNode))
29: write($M[node.leftChild_{id}]$)
30: write($M[node.rightChild_{id}]$)
31: **end if**
32: **end while**
33: **end function**

with the correct offset values. The `KDTreeNode` structure holds the partitioning plane, memory pointers to the left and right child nodes, and a pointer to a list of triangles, valid only for leaf nodes. The out-of-core equivalent, `KDTreeFlatNode`, replaces the memory pointers with node offsets from the root, the latter's offset being zero. Moreover, the structure does not contain information about any

geometry held at a node except for the number of primitives. When a leaf node is persisted (lines 10-15), geometry primitives are indexed in a following block, with the respective memory pointers changed to offsets from the start of the triangle list t (which is independently persisted). During the first traversal, a map $M : node_{id} \mapsto offset$ is incrementally updated to keep track of the offset of each node in the contiguous buffer. The tree is then traversed a second time to update the left and right child nodes of each of the persisted nodes to point to the correct offsets (lines 23-32).

5.5 Rendering in RaaS

The rendering process is encapsulated in the *compute* states of the Task Pipeline (Figure 5.3). Problem decomposition and task distribution among the participating resources are largely independent of the framework within which they operate. These resources are provided with an environment for computation which utilises communication channels independent from those used for system control and management.

5.5.1 Work Distribution

The system is designed to support various work distribution strategies, as long as results are returned by the Task Coordinator. The model currently utilised is based on the Master-Worker paradigm. In general, a master entity decomposes a problem into a number of independent sub-problems or tasks (bag of tasks) and makes them available to workers. The latter pick up these tasks, perform the respective computation and return the results. This process is repeated until all tasks have been completed, at which point the results are merged by the master.

The frame scheduled for rendering is partitioned into a number of tiles, which may be of variable sizes, even within the same frame. The partitioning scheme is agreed upon by both master and workers, such that each tile can be uniquely identified by an index. The index can mark a continuous region or act as a seed for a low-discrepancy sequence (Van der Corput, 1936; Sobol, 1994), in which case the elements within the tile are staggered over the frame (Aggarwal *et al.*, 2009). The master sends a single tile index to each of the workers in the group. Having rendered the respective tile, a worker sends the results back to the master and asks for more work. The master then responds by sending a new tile or a

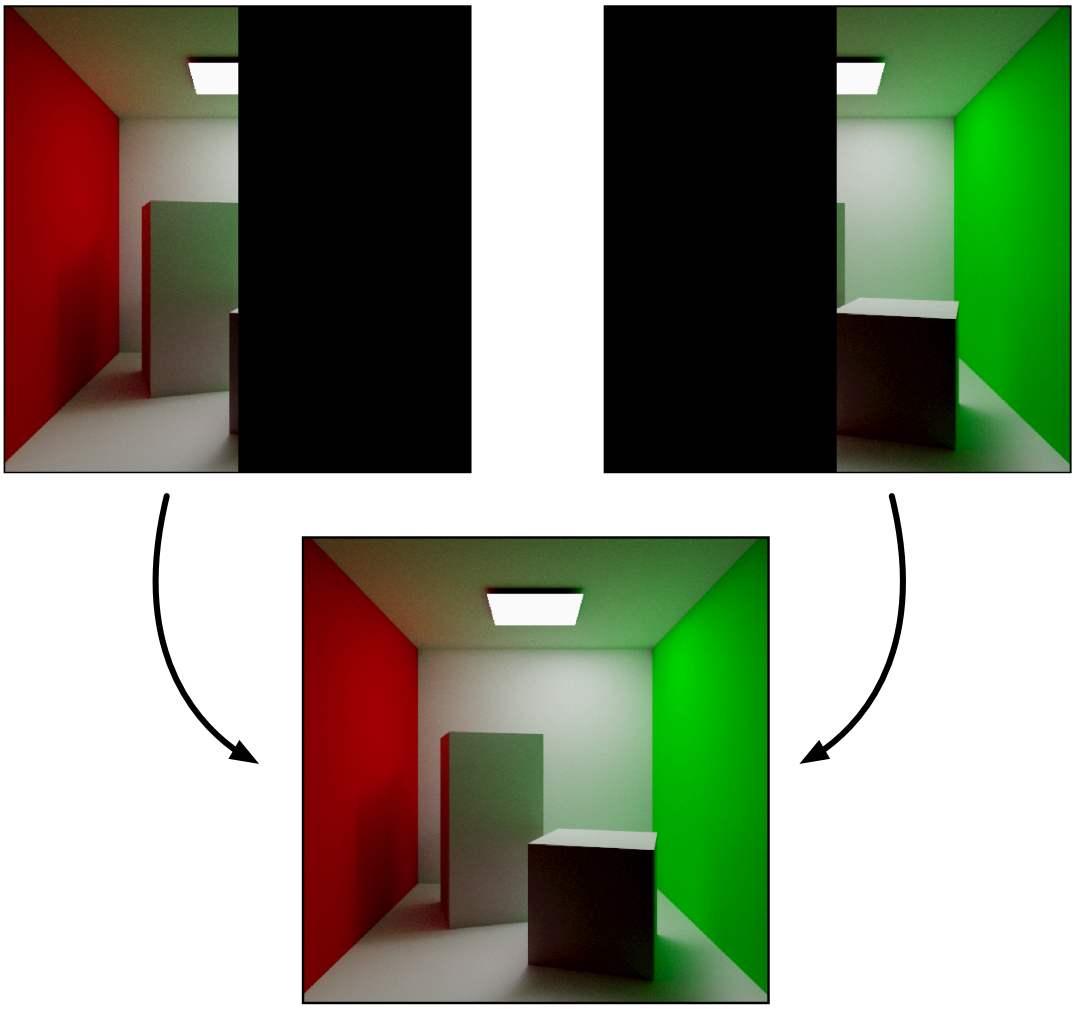


Figure 5.4: In region-based work distribution, each tile represents a spatially coherent region over the image frame that is disjoint from all other regions.

termination signal when the work queue has been exhausted, marking the end of the computation phase. If a worker does not respond within a stipulated time period, the master reschedules the corresponding computation.

Initially, each tile distributed to a worker resource marks a contiguous region (Figure 5.4); if the master-worker set consistently fails to render a frame within any time constraints specified in its job description, then the system switches to a frameless approach, with each tile representing a series of pixels drawn from a low discrepancy distribution over the whole frame (Figure 5.5). When the frameless approach is engaged, the Task Pipeline interactions between task coordinator and worker (Figure 5.3) change slightly to decouple the synchronisation and the computation stages and allow them to run concurrently.

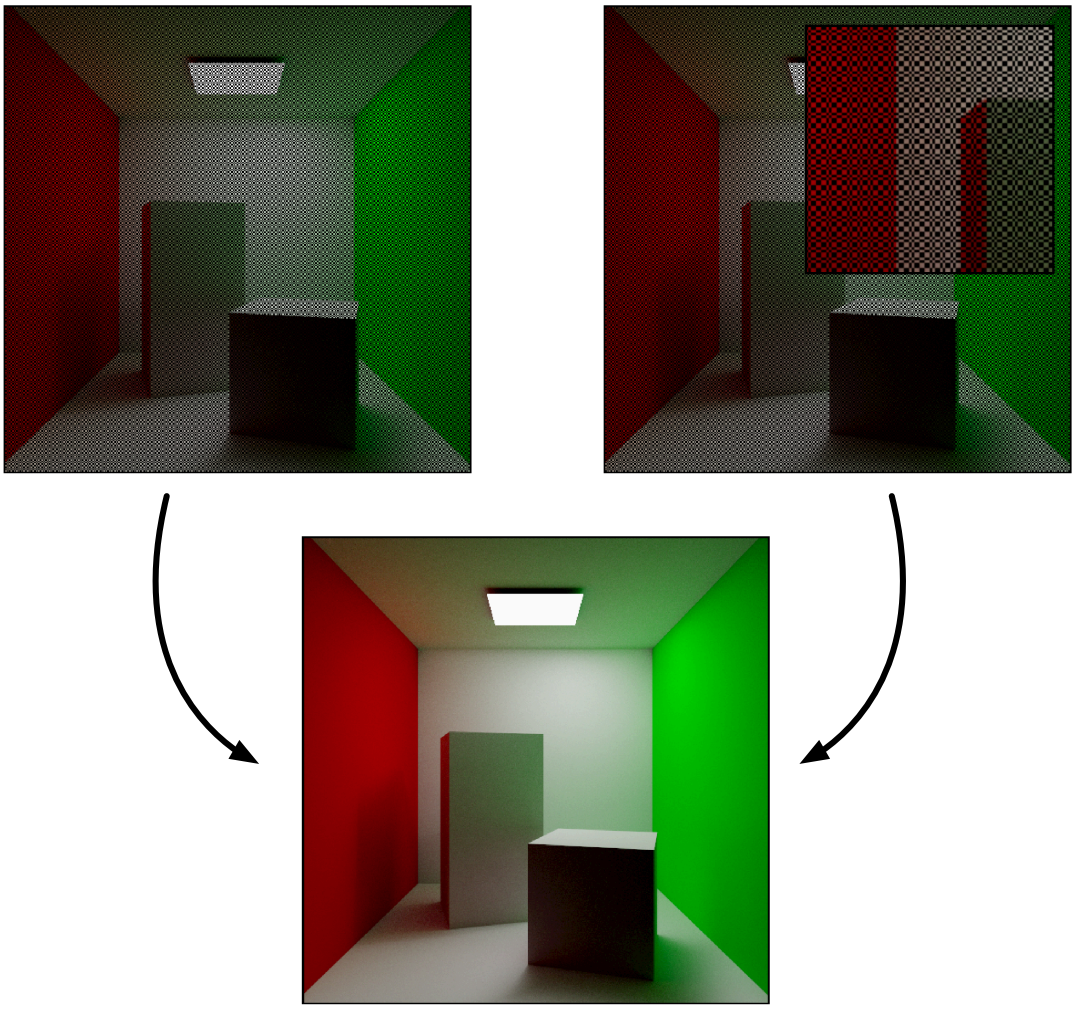


Figure 5.5: In the frameless approach, each tile represents a sequence of pixels over the whole image frame, staggered using a low-discrepancy sequence. The regions bounding the tiles are not disjoint.

5.5.2 Communication

Communication overheads for interactive rendering may be substantial, swamping computation time due to the large and frequent data transfers that occur between workers and master (Task Coordinator). An uncompressed high-definition image may well occupy 6 MB of memory (1920×1080, 24-bit colour), or considerably more if HDR is taken into account; the bandwidth requirements for interactive rendering are very demanding on the interconnect infrastructure since within a single second, multiple images (frames) must be transferred. This is further exacerbated by techniques that require passing geometry buffers (normals, direct and indirect lighting contributions, albedo, etc) between workers and Task

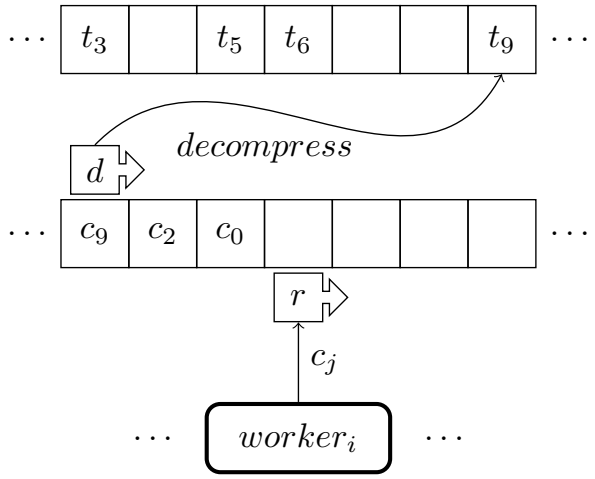


Figure 5.6: Tiles received from worker processes are stored in a staging buffer and asynchronously decompressed to prevent a bottleneck forming at the Task Coordinator. The marker r points to the next empty buffer, while d points to the last decompressed tile. Processed tiles are copied to their respective position in the frame buffer (e.g. c_9 goes to t_9).

Coordinator. In order to alleviate the effects of communication overheads and bandwidth limitations, all results sent back to the master are compressed using a lossless scheme.

In a straightforward approach, the master would decompress the results as soon as they come in, but this would introduce a bottleneck, forcing the workers to wait for their next task more than is necessary. Moreover, if the workers return all at once, performance would degrade further due to the contention introduced at the master. To minimise worker delays, a slightly different approach was taken, where decompression of results and frame composition were decoupled from the receiving thread. Results are received into a circular buffer and a decompression thread asynchronously expands and orders them into a frame buffer (Figure 5.6). This decoupling allows the master to respond to workers' requests for work more quickly.

5.5.3 Post-processing Filters

RaaS provides a number of post-processing techniques with the aim of improving the quality of the final result. Generally these filters either have system-wide application and can be applied to synthesised frames irrespective of which rendering algorithm was used to generate them (e.g. tone mapping), or are tied to a

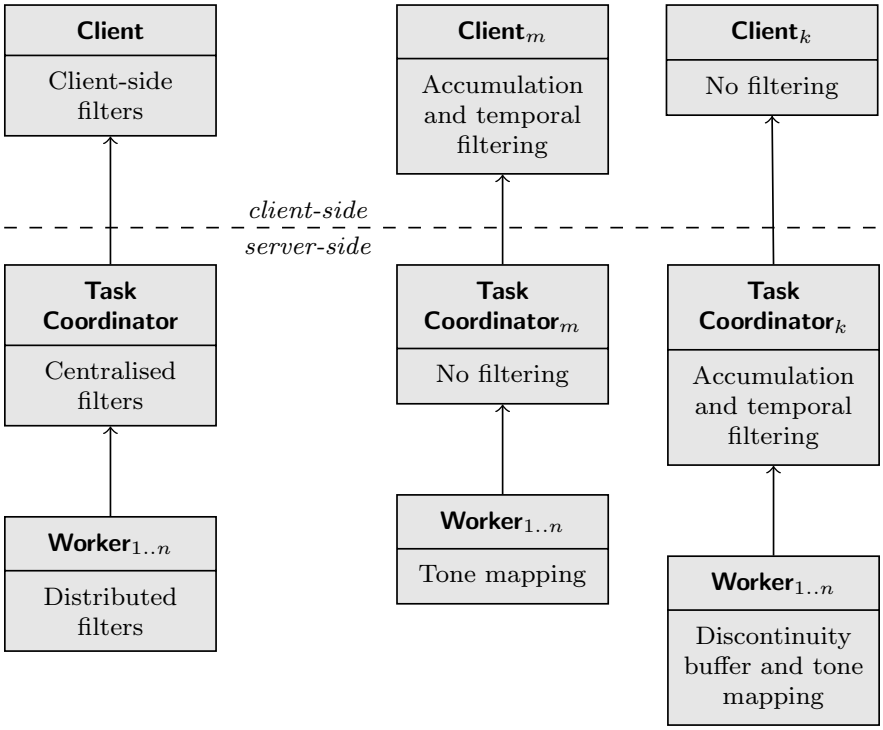


Figure 5.7: Three post-processing filter configurations are shown; the left configuration is generic, specifying which kind of filter goes where. For instance, distributed filters can execute at a worker process, while centralised filters execute on the Task Coordinator. Centralised filters may also execute at the client, but whether this is feasible or not depends on the communication required to satisfy the data dependencies of the filter. The configurations on the right exemplify two typical scenarios where filters are applied at different stages.

specific rendering algorithm (e.g. discontinuity buffer for interleaved sampling). Either way, filters are applied to a frame before it is presented to the user. In the framework, the application of post-processing filters is categorised into two classes: *distributed* and *centralised* (Figure. 5.7). Distributed filters are usually local filters and do not require a complete view of the image or additional associated buffers. Workers can apply distributed filters on their assigned tile, after radiance has been computed for the region. The tiles are then assembled at the Task Coordinator. On the other hand, centralised filters require information that is not entirely available to any single worker. For instance, in the case of the application of a spatial filter such as box blur, the filter iterates through each pixel in the image and replaces its colour with the average of the neighbouring pixels. The neighbourhood centred about each processed pixel is called the *window*, or *kernel*. If applied as a distributed filter, the kernel must be truncated at the

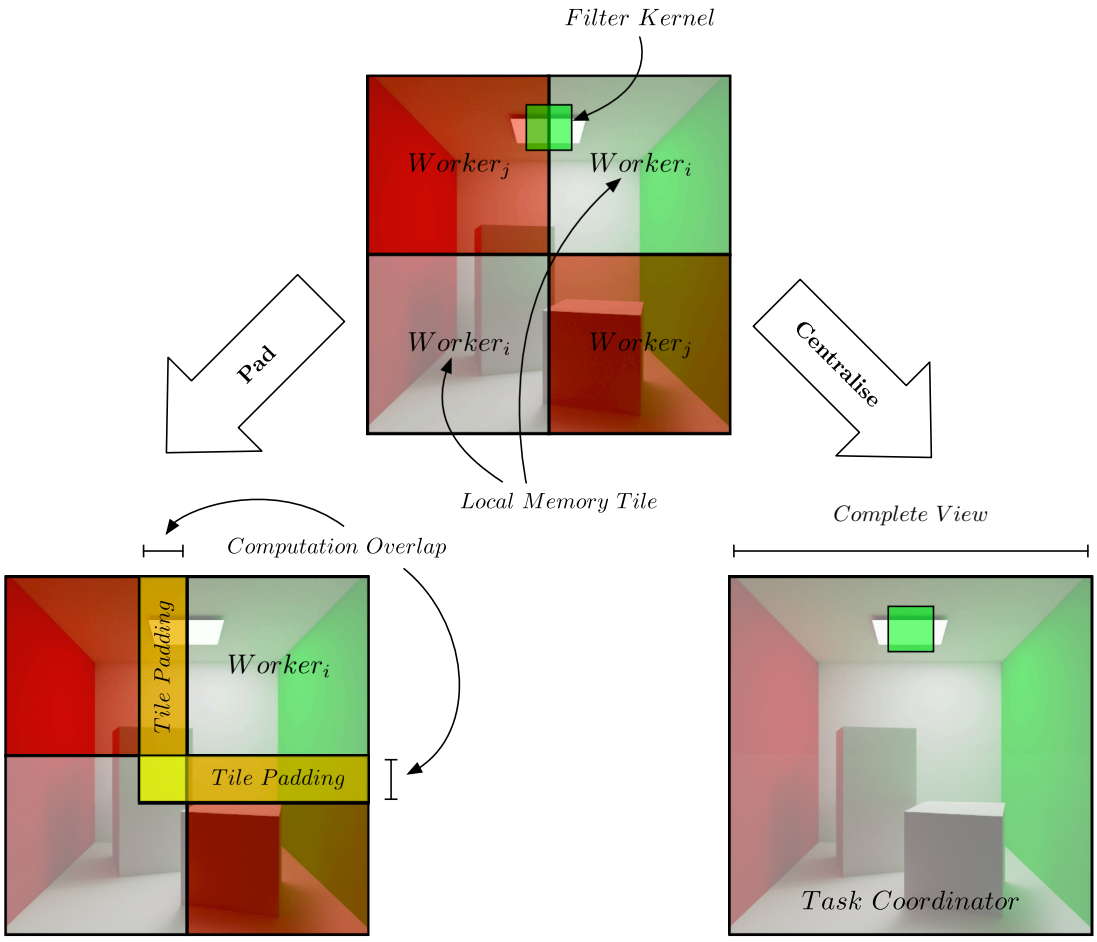
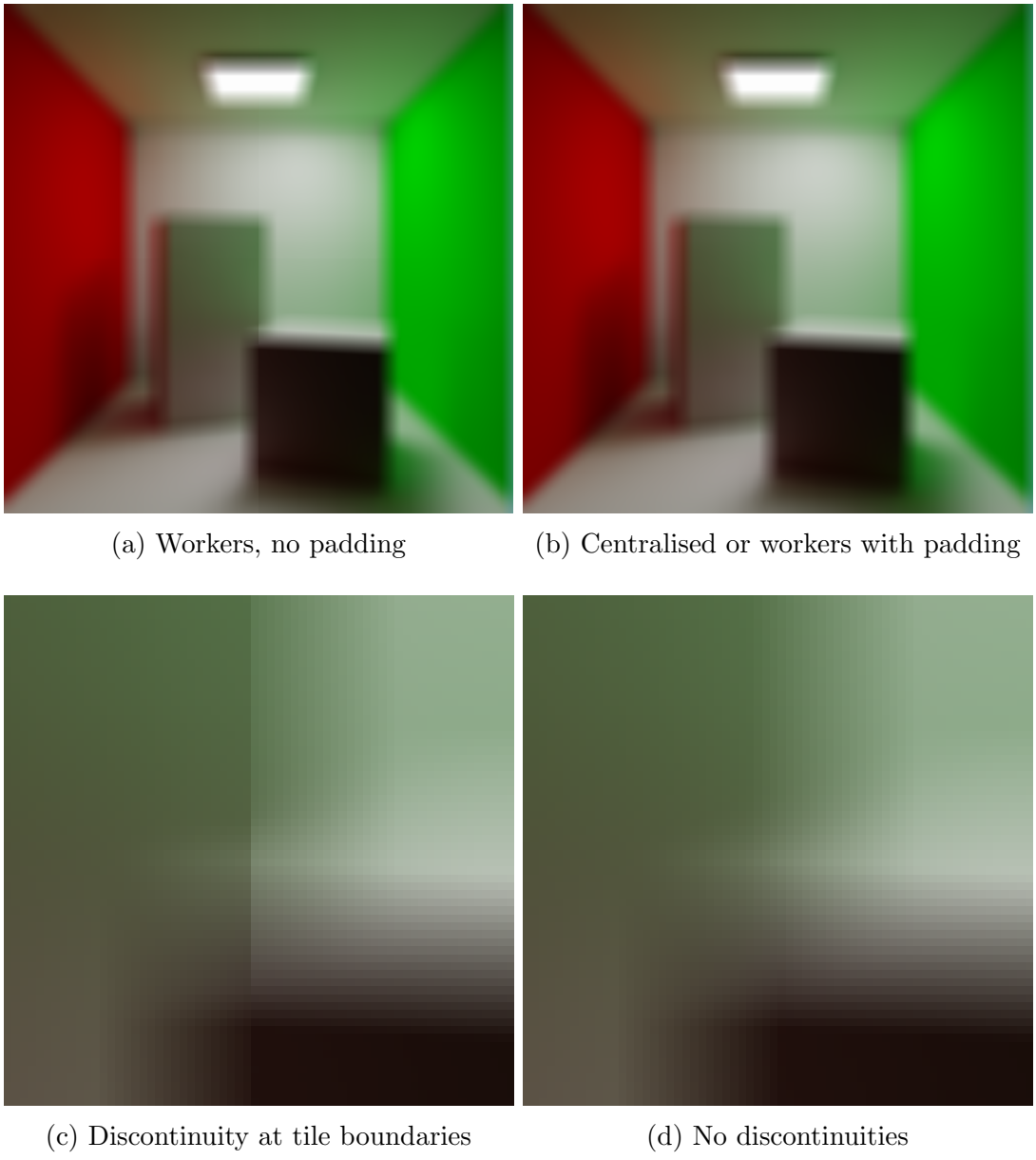


Figure 5.8: Centralised filters require more information than what is available at any individual worker. Image blurring is a typical example of a centralised filter; here a kernel is convolved with the image to achieve the blur effect and cannot be applied as a distributed filter at the individual workers because it lacks a complete view of the image. In cases where the communication overhead associated with aggregating the required data at a Task Coordinator is overly high, the regions marked by tiles can be padded to encompass any required additional information from adjacent tiles, albeit at the cost of duplicating computation.

boundaries of each tile, thus failing to account for logically adjacent pixels that would have fallen within the neighbourhood, had the entire image been processed instead. This introduces artefacts and discontinuities at tile boundaries when the tiles are composited into the final image, as shown in Figure 5.9.

The use of centralised filters prevents the computation from being amortised over the workers, introducing an additional sequential component to the rendering pipeline and a loss of scalability of the system in general. Moreover, communication between workers and master is also increased in cases where rich



(a) Workers, no padding

(b) Centralised or workers with padding

(c) Discontinuity at tile boundaries

(d) No discontinuities

Figure 5.9: Artefacts from executing centralised filters at workers without padding tile boundaries. Figure 5.9a, shows the result of applying a blurring filter at the workers without the necessary padding. Figure 5.9c provides a detail view of the seam running down the middle part of the image, where the horizontal tile boundaries lie. Figures 5.9b and 5.9d show the corresponding results when running the blur as a centralised filter or as a distributed filter on padded tiles.

buffers are required for filtering, limiting system bandwidth. This is the case with geometry-aware post-processing filters which may require depth and normal buffers, together with the colour buffers, for instance. Conversely, distributed fil-

ters minimise communication overheads as only final colour values need be sent to the master process. Notwithstanding the constraints over distributed filters and their local nature, a tradeoff can be made when the communication overheads of a centralised filter are too steep, and execute the filter by the workers. Under the proviso that additional information required by the filter is available in spatially adjacent tiles, the region is enlarged, or padded, to encompass this information, forcing the system to render additional pixels at the borders of a tile. The filter is then applied on the padded tiles, and the padding discarded once the post-processing work is complete. The tile sent back to the master is free of discontinuities and artefacts. Unfortunately, the padding results in computation overlap, or overdraw, which grows with the number of tasks per frame; as emphasised, this approach is only viable when the loss in scalability or communication costs become excessive. Currently, the only way of determining the performance of one approach over the other is via empirical tests.

5.5.4 Tone Mapping

Tone mapping is performed on resultant frames, to facilitate the transformation of high dynamic colour ranges into 8-bit RGB triples. The system supports a number of techniques of varying quality and complexity (Schlick, 1995; Durand & Dorsey, 2002; Drago *et al.*, 2003). The amenability of a technique to parallelisation is an important factor in deciding where it should be applied (i.e., at what stage and by whom, Figure 5.7 *server-side*). Thus, if a technique does not require access to regions of the image besides what is already available at a worker (e.g. global tone mapper), it can be applied using a *distributed filter* before the region is compressed and sent back to the Task Coordinator; otherwise it has to be applied at the Task Coordinator using a *centralised filter*.

5.5.5 Progressive Rendering and Temporal Filtering

Interactive rendering imposes demands on frame generation times that limit the quality of global illumination solutions. Progressive rendering is used to amortise the cost of computing a high quality solution over a number of successive frames, when the scene and observer are static. Conversely, when the state of the scene or observer changes, the temporal information in successive frames is used to reduce artefacts due to discontinuities between progressively accumulated frames

and newly generated ones. In particular, the system can perform filtering on a rendered frame P_f prior to presenting it to the user by availing itself of two further contributing signals: an accumulation contribution P_a and a temporal contribution P_t.

The accumulation contribution results from progressive rendering of the scene when both observer and the scene itself do not express any changes. The state of the scene is encoded in a vector $\mathbf{S}$ and changes between a frame n and the next $(n+1)$ are captured by the expression $\delta(\mathbf{S}_{n+1} - \mathbf{S}_n)$, where a value of zero represents a change in consecutive frames, and one otherwise. The accumulation component for frame $n+1$, $P_a[n+1]$, is computed by aggregating the accumulation component for the previous frame $P_a[n]$ with P_f, weighted by the change in the scene:

$$P_a[n+1] = \delta(\mathbf{S}_{n+1} - \mathbf{S}_n) \cdot (P_a[n] + P_f). \tag{5.4}$$

It follows that whenever the scene state changes between a frame and the next, the accumulation contribution is reset to zero. This may result in abrupt discontinuities in animation quality. The aim of temporal filtering is that of removing visual artefacts and flickering between consequent frames in the animation; the temporal contribution is the combination of the last frame rendered and the temporal contribution history (or the temporal contribution at the previous frame), weighted by w_f and w_t, for which typical values are $w_f = 0.6$ and $w_t = 0.4$ respectively.

$$P_t[n+1] = w_t P_t[n] + w_f P_f \tag{5.5}$$

When computing a filtered frame, the temporal contribution P_t is predominant if the scene or observer state are changing $[\delta(\mathbf{S}_{n+1} - \mathbf{S}_n) = 0]$; conversely, if the state is unchanged from previous frames, the result will transition towards the progressive rendering contribution P_a. The transition from one contribution to the other is dependent on the transition length t_{max} and is expressed in number of frames. The final filtered frame presented to the user is thus computed:

$$P_p = \begin{cases} \dfrac{P_a[n]}{count} & t = 0 \\ \dfrac{P_a[n] + P_t[n]}{count + 1} & 0 \leq t \leq t_{max} \\ P_t[n] & t = t_{max} \end{cases}, \tag{5.6}$$

where *count* is the number of frames progressively rendered and accumulated in

P_a, and t represents the frame transition. Whenever a change in scene state is captured, t is initialised to t_{max}; when the scene state is unchanged, t decreases monotonically until it is equal to zero ($t = 0$).

5.5.6 Client-side Post-processing

Computations which cannot be carried out by workers translate into a sequential component which limits scalability. Thus, the kind of computations carried out at the Task Coordinator are limited and performed centrally only if the costs of distributing them would outweigh the benefits of parallel computation. The penalty incurred by such computations is especially evident at high frame rates, where a sequential computation time of 100 ms is enough to limit the maximum frame rate to 10 Hz. To overcome this potential bottleneck, computations which do not incur additional communication overheads can be offloaded onto the client, provided the latter has enough computing power (Figure 5.7, *client-side*). In practice, offloading a process like accumulation and temporal filtering onto a client requires no additional information be transmitted except notifying the client that it has to perform the computation itself. On the other hand, a process such as the discontinuity buffer requires substantially more information to be transmitted; the direct and indirect lighting contribution have to be stored separately, since they are merged only after the process has completed, and depth information is required to perform geometry-aware smoothing. Streaming this information to the client substantially increases the bandwidth requirements of the system (see §3.5.3).

5.5.7 Rendering Techniques

Flexibility is very important when offering rendering as a service, and thus, a frame can be synthesised using a variety of methods. The system currently supports a number of techniques spanning the fidelity spectrum, from Whitted-style ray tracing (Whitted, 1980) to Path Tracing (Kajiya, 1986). The selection of a particular rendering algorithm is hugely dependent on user preference. A user looking for high interactivity at the cost of realism might go for Whitted-style ray tracing, while another looking for improved graphical fidelity at the cost of reduced interactivity might opt for Path Tracing, assuming the number of dedicated resources are the same in both cases. There are also other techniques

which offer a better speed-quality compromise at the cost of a less straightforward parallel implementation, one such technique being Instant Global Illumination (IGI) (Benthin *et al.*, 2003).

Our reference implementation of IGI is not optimal as it does not trace rays in packets nor take advantage of SIMD instructions (Benthin *et al.*, 2003), and lacks features like reordering of computations and optimised layout and alignment of data structures (Wald, Benthin & Slusallek, 2002). Thus, we have implemented a lower quality variant, IGI-X, which subsamples the indirect lighting component, resulting in a low resolution indirect lighting buffer that has to be upscaled before being merged with the direct lighting component. Upscaling is performed either via bilinear filtering or via edge-preserving bilateral filtering (Sloan *et al.*, 2007), depending on whether priority lies in speed or quality. Users are not constrained to a single rendering algorithm or filtering strategy per job, but are able to switch techniques on the fly.

(a) Sponza Palace (Crytek)

(b) Kalabsha Temple

(c) Conference Room

(d) Šibenik Cathedral

Figure 5.10: The scenes used for evaluating the implementation of Rendering as a Service.

5.6 Results

The system is evaluated on the basis of its scalability and elasticity; the setup consist of a supercomputing cluster of 64 processing nodes, with 12 cores at each node (2×Six-Core AMD Opteron Processor 2431), 2 TB of main memory (32 GB per node), and Infiniband DDR Interconnect. Each node is also connected to shared secondary storage via Networked File System (NFS). Distributed memory interprocess communication within the cloud system is carried out using the Message Passing Interface (MPI) (Walker & Dongarra, 1996), specifically, MPICH2 (version 1.1.1p1). Client control communication employs connection-oriented sockets, while output streaming utilises the Real-time Transport Protocol (RTP) (Jacobson *et al.*, 2003). The supercomputing cluster was not exclusively used by our system; 144 cores were allocated across all of the 64 nodes. Four test scenes were used in the evaluation of the system: the Atrium of Sponza Palace, remodelled by Frank Meinl at Crytek (Figure 5.10a, 262K faces, 153K vertices), the temple of Kalabsha from Sundstedt *et al.* (2004) (Figure 5.10b, 860K faces, 438K vertices), the conference room (Figure 5.10c, 331K faces, 189K vertices), and Šibenik cathedral (Figure 5.10d, 75K faces, 41K vertices). The scenes were held on a central repository accessible via Gigabit Ethernet (see §5.4.3). Rendering was performed at resolutions of 512×512 and 1024×1024, with super sampling disabled. Refinement occurred by means of progressive rendering and accumulation between sequential frames (§5.5.5). For IGI, a 3×3 interleaved sampling pattern was used. The number of virtual point lights (VPLs) per scene was set to 32. VPL shooting was performed by the workers at every frame; seeds for the pseudo-random sequences used in the generation were made consistent during the synchronisation step. A discontinuity buffer was used to efficiently reduce the variance in irradiance introduced by interleaved sampling; a 3×3 kernel was used to filter noise. This post-process was carried out at the workers, as a distributed filter, and an additional 1-pixel border was computed for each tile (Benthin *et al.*, 2003). Frameless rendering has been disabled for the recording of results, due to its incompatibility with some of the rendering methods used.

5.6.1 Rendering Scalability

The capability of the system to scale resources during the rendering of a single job is first evaluated, with the aim of showing that adapting traditionally paral-

Scene	Resolution	Path Tracing	IGI	Whitted
Conference	512×512	0.54	0.29	3.53
Kalabsha		0.78	0.47	4.42
Šibenik		0.70	0.44	4.13
Sponza		0.68	0.41	4.03
Conference	1024×1024	0.07	0.13	0.88
Kalabsha		0.19	0.12	1.13
Šibenik		0.17	0.11	1.03
Sponza		0.17	0.10	1.00

Table 5.1: The table shows the reference times for dedicated shared memory rendering of the scenes without cloud overheads, on a 3.2 GHz quad-core Intel Core i7-960 processor. The figures are given in frames per second (FPS) for three different rendering techniques, Path Tracing, IGI and Whitted-style ray tracing. IGI-X has not been recorded since a shared memory implementation was not available.

lel rendering algorithms into the system has a minimal impact on their expected scalability, even with the additional scheduling and resource management overhead of rendering as a service. To place scalability results in context, Table 5.1 lists the rendering performance, in frames per second, of the scenes used for the evaluation when run on a dedicated shared memory multithreaded system using a special implementation that is bereft of any of the associated job scheduling and management overheads. Another advantage of the standalone shared memory implementation is that output goes directly into a frame buffer mapped to the display, rather than being encoded and streamed over the network to the client process.

In figures 5.11, 5.12, 5.13 and 5.14, the speed-up curves show the performance of the employed rendering algorithms for all the tested scenes over an increasing number of workers. There is a linear speed-up in the rendering for up to 16 workers, after which, the speed-up becomes sublinear, with efficiency decreasing as more workers are added. When contrasting the different problem sizes for the same rendering algorithm (512×512 versus 1024×1024), it can be observed that for larger problem sizes, the system is more efficient, resulting in better scalability. To express this in terms of Amdahl's Law (see 4.1), for smaller problem sizes, as the number of workers increases and the parallel computation per processor decreases, per frame synchronisation and communication overhead, which constitute a sequential bottleneck, become the dominant term in the speed-up

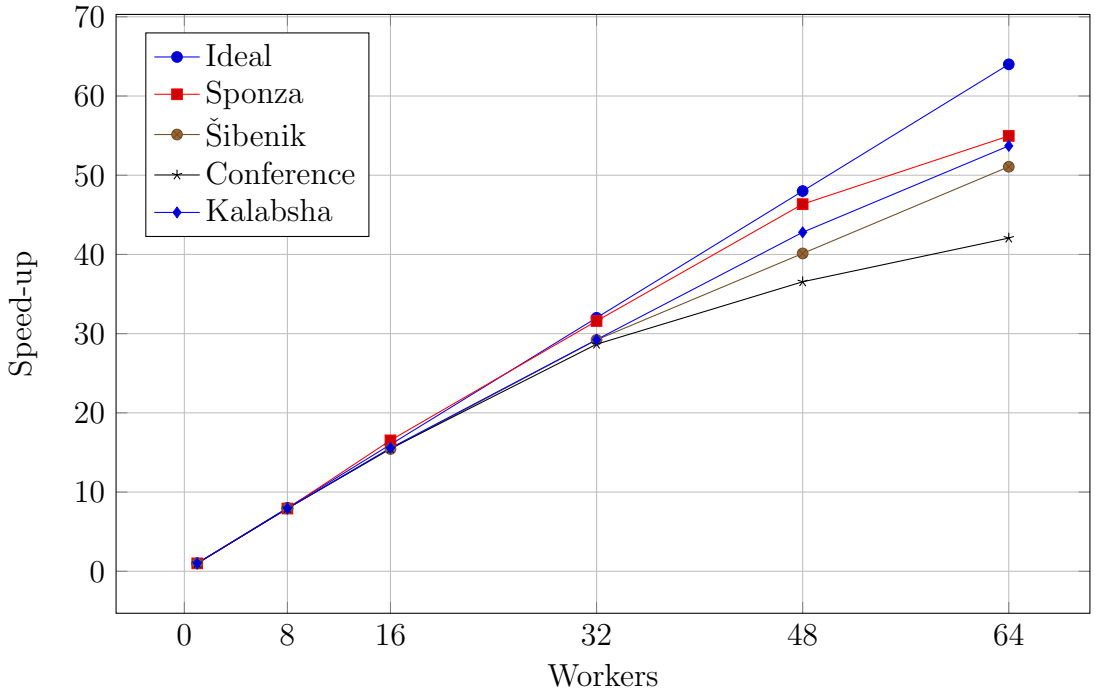
computation, effectively limiting the scalability of the system. Increasing the problem size, by increasing the output resolution, provides more work per frame to the increasing number of workers, resulting in a larger parallel to sequential computation ratio, and thus, more efficient scalability, as evidenced by figures 5.11b, 5.12b, 5.13b and 5.14b.

Figures 5.15 and 5.16 provide insight into the performance and scalability of the individual rendering algorithms. In particular, it can be observed that even for similar problem sizes (in terms of the number of pixels), the scalability of the different rendering algorithms diverges greatly beyond 32 workers. The algorithms that yield faster frame rates per worker experience a larger speed-up degradation at higher worker counts (> 32), as shown in figures 5.13 and 5.14. This degradation seems to correlate inversely with the bandwidth usage, and communication in general. Faster rendering algorithms generate more frames per second, and thus require more frequent data transfers between the master and the workers. Tasks are composed of 32×32 pixel tiles, and for colour buffer transfers this would amount to 3 KB per tile, or 768 KB per frame. Ignoring any compression, the bandwidth requirements for a job running at 30 Hz approach 180 Mbps. Although the interconnect employed in the testbed can handle this kind of throughput, the lack of message batching and the contention introduced via communication with a single master process contribute to the observed degradation. Furthermore, the synchronisation window (see §5.4.2) contributes to the frame setup time, and although it is bounded at 5 ms, it is still a sequential component that impinges on the scalability of the system.

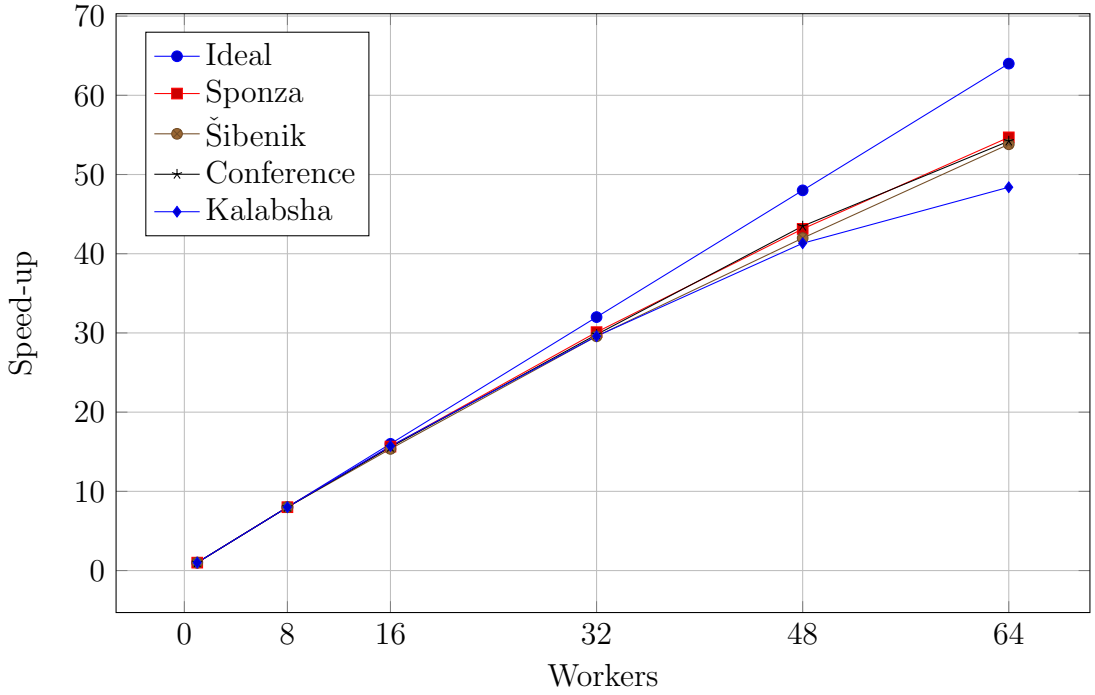
5.6.2 Client Scalability (Cloud System Overhead)

The management of multiple clients connected to the system requires jobs to run over an additional layer of resource scheduling and management. In order for the system to be a viable alternative to single-client dedicated rendering setups, this overhead should be as low as possible. Two facets of system overheads are assessed, namely, how multiple concurrent clients affect one another, and the magnitude of the realisation penalties incurred per frame when compared to a dedicated rendering system.

To evaluate the effect of multiple concurrent clients on the rendering performance, a number of jobs were introduced to the system at different arrival times and their output performance was measured. In particular, three jobs J_0, J_1 and

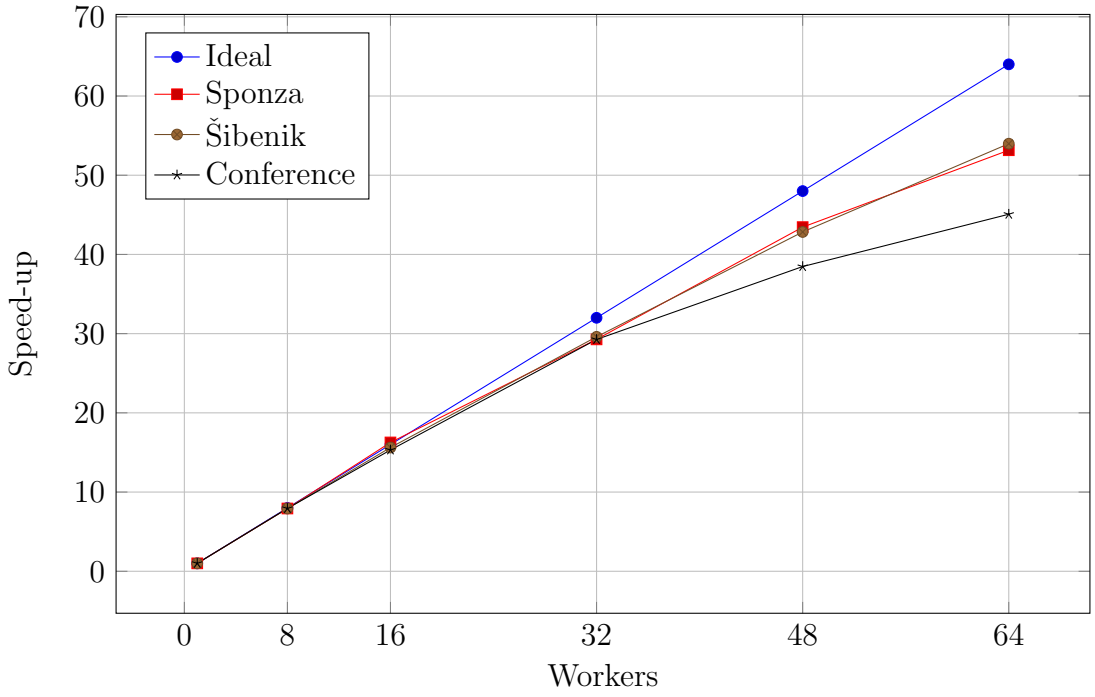


(a) Scalability results for all scenes rendered using Path Tracing at a display resolution of 512 × 512, for up to 64 workers.

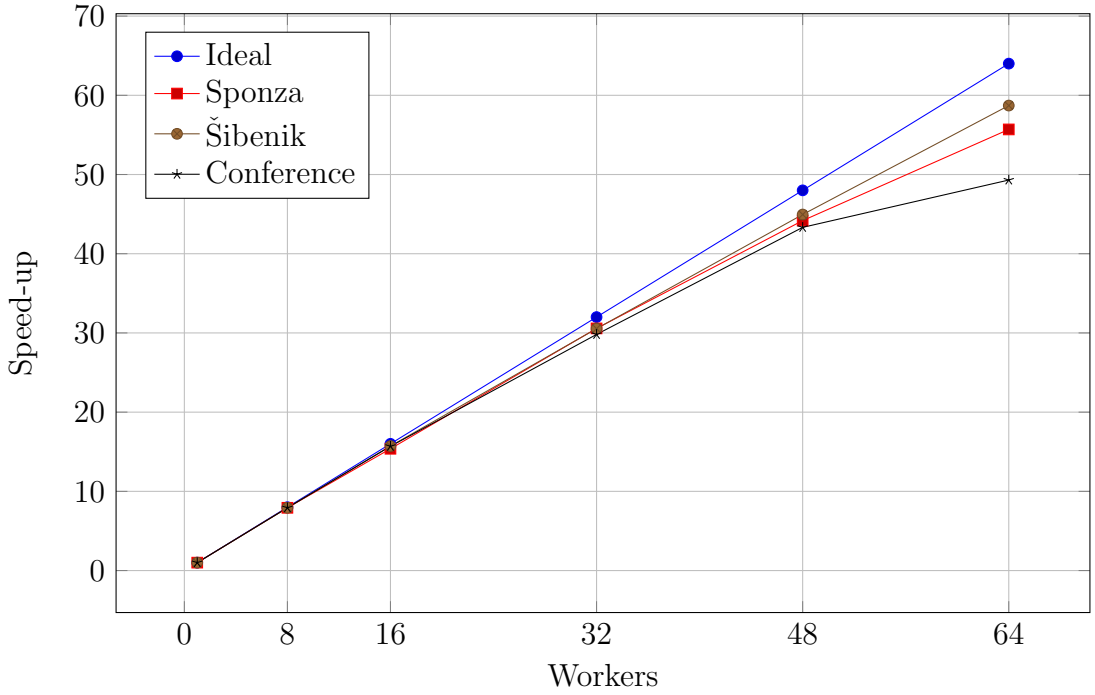


(b) Scalability results for all scenes rendered using Path Tracing at a display resolution of 1024 × 1024, for up to 64 workers.

Figure 5.11: Scalability results for Path Tracing.

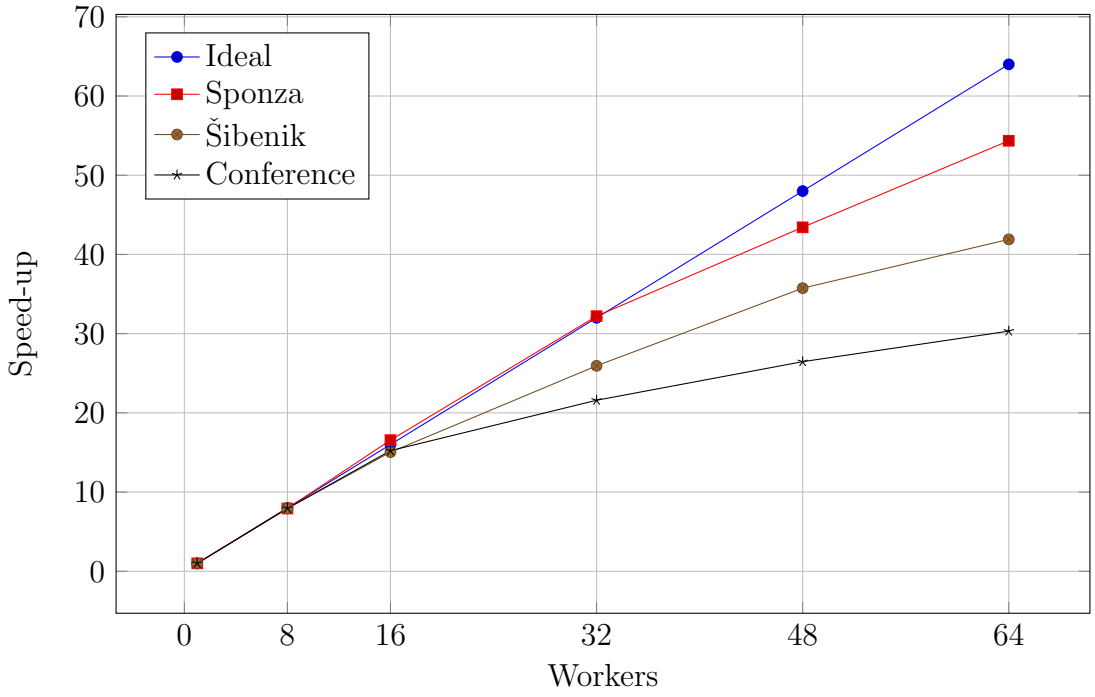


(a) Scalability results for all scenes rendered using IGI at a display resolution of 512 × 512, for up to 64 workers.

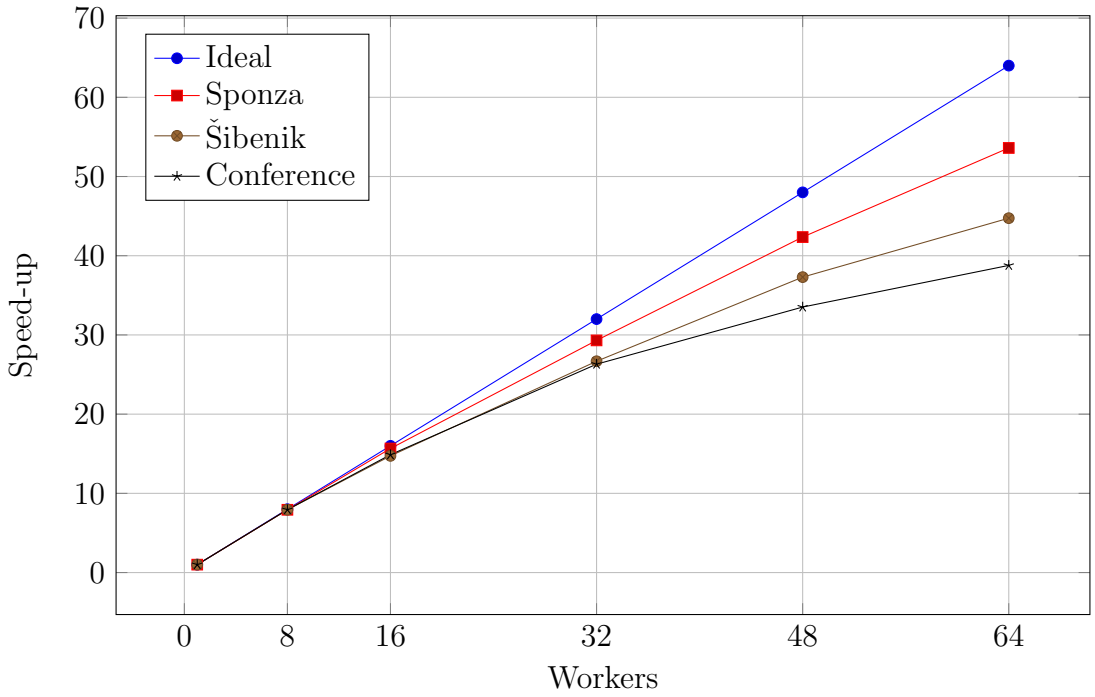


(b) Scalability results for all scenes rendered using IGI at a display resolution of 1024 × 1024, for up to 64 workers.

Figure 5.12: Scalability results for IGI.

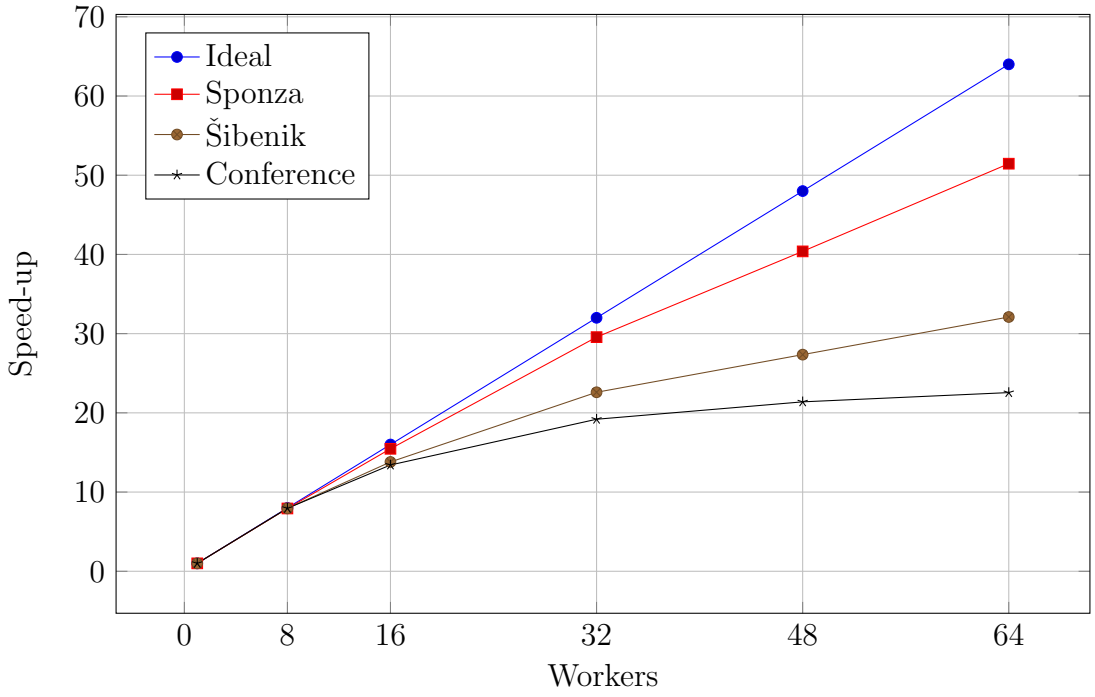


(a) Scalability results for all scenes rendered using upsampled IGI-X at a display resolution of 512 × 512, for up to 64 workers.

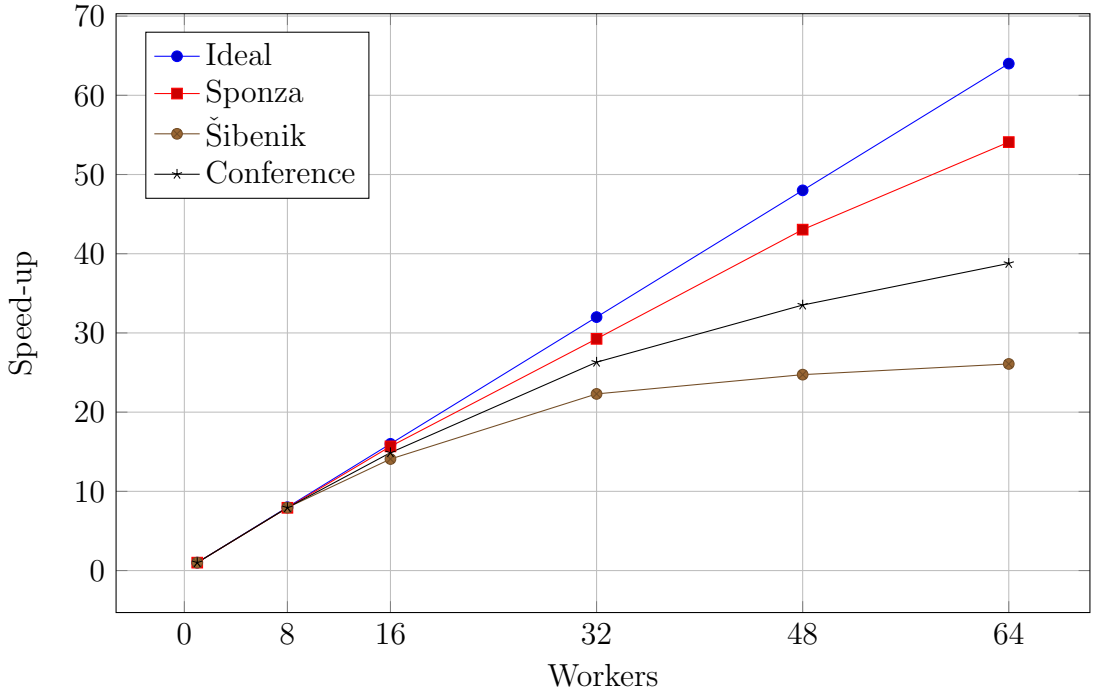


(b) Scalability results for all scenes rendered using upsampled IGI-X at a display resolution of 1024 × 1024, for up to 64 workers.

Figure 5.13: Scalability results for upsampled IGI-X.

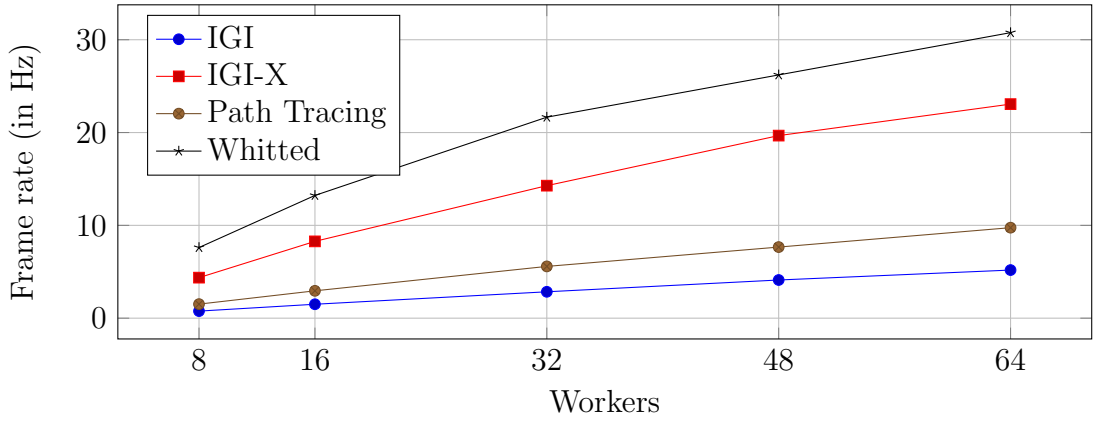


(a) Scalability results for all scenes rendered using Whitted-style ray tracing at a display resolution of 512 × 512, for up to 64 workers.

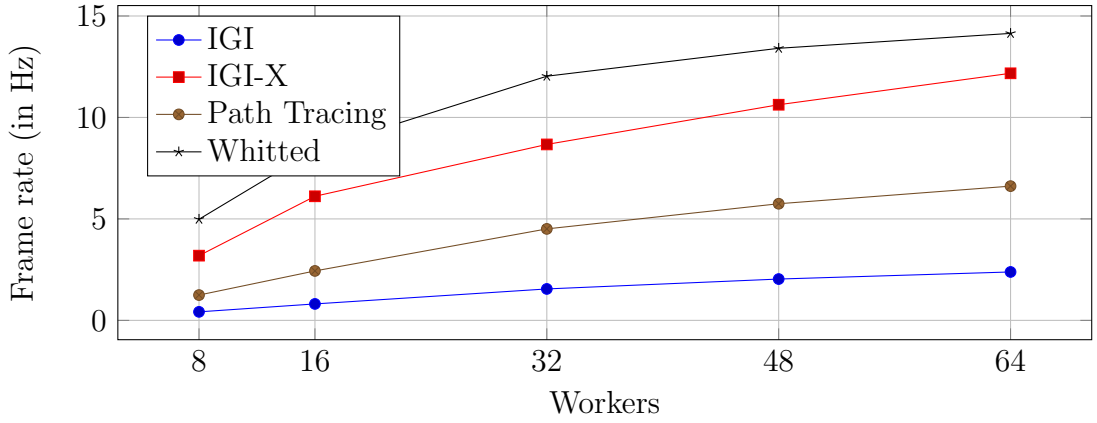


(b) Scalability results for all scenes rendered using Whitted-style ray tracing at a display resolution of 1024 × 1024, for up to 64 workers.

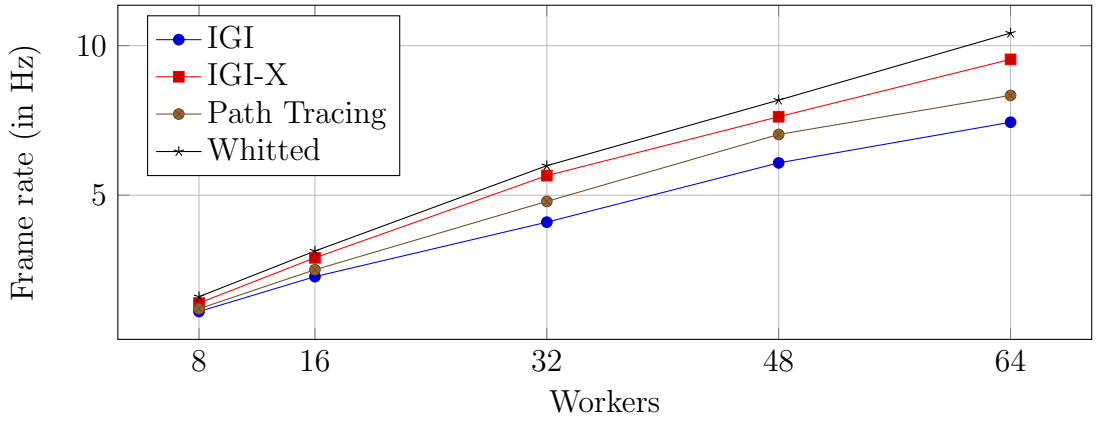
Figure 5.14: Scalability results for Whitted-style ray tracing.



(a) Šibenik Cathedral

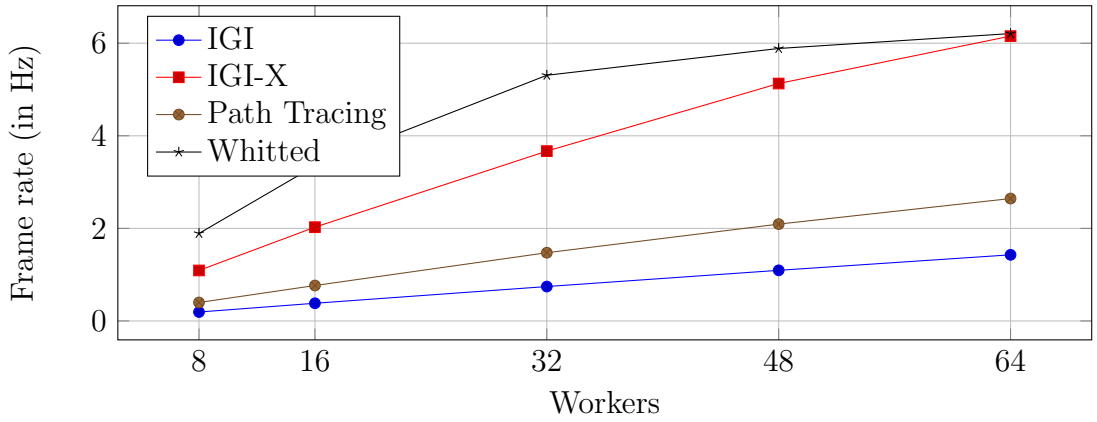


(b) Conference Room

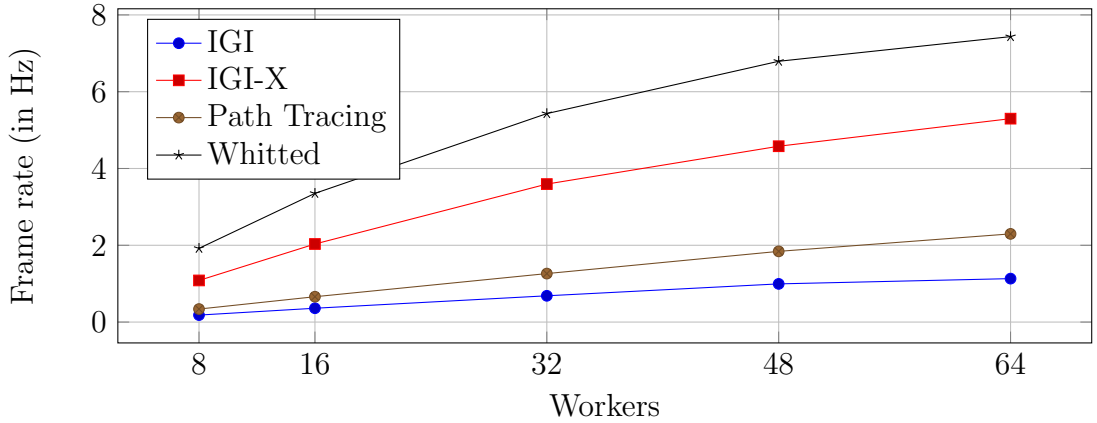


(c) Sponza Atrium

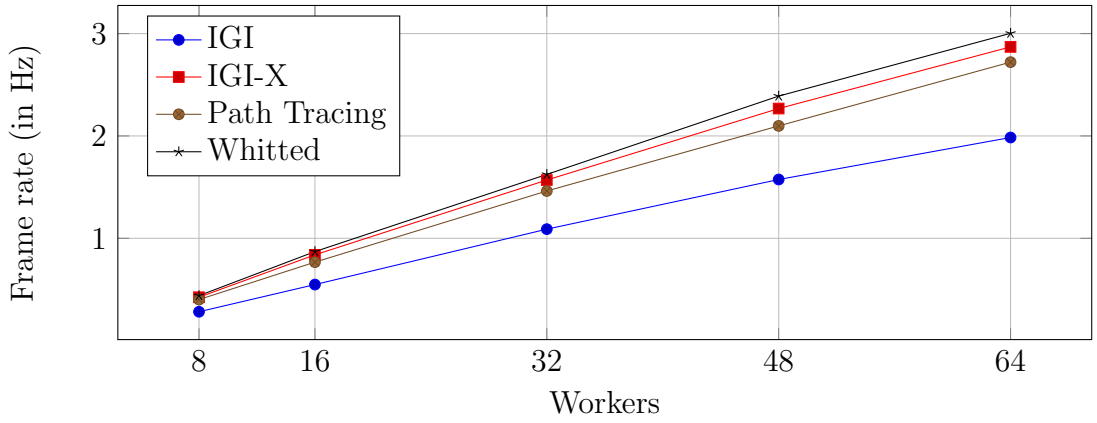
Figure 5.15: Rendering performance for different worker configurations at 512 × 512. Whitted-style ray tracing performs best with respect to frame rate, followed by IGI-X. However, the results for these two algorithms suggest that the communication and synchronisation overheads may be swamping computation as the number of processors increases.



(a) Šibenik Cathedral



(b) Conference Room



(c) Sponza Atrium

Figure 5.16: Rendering performance at 1024 × 1024.

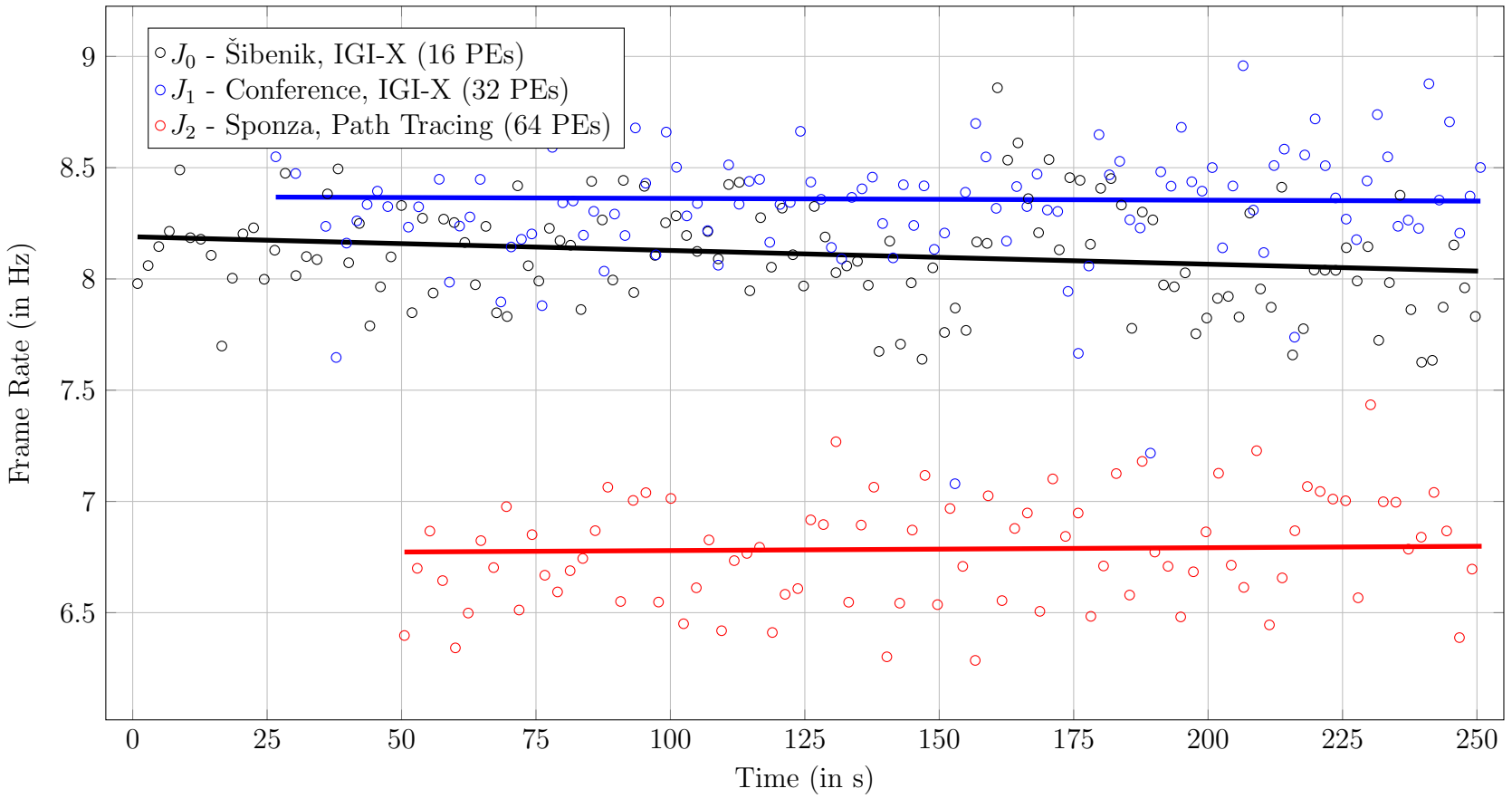
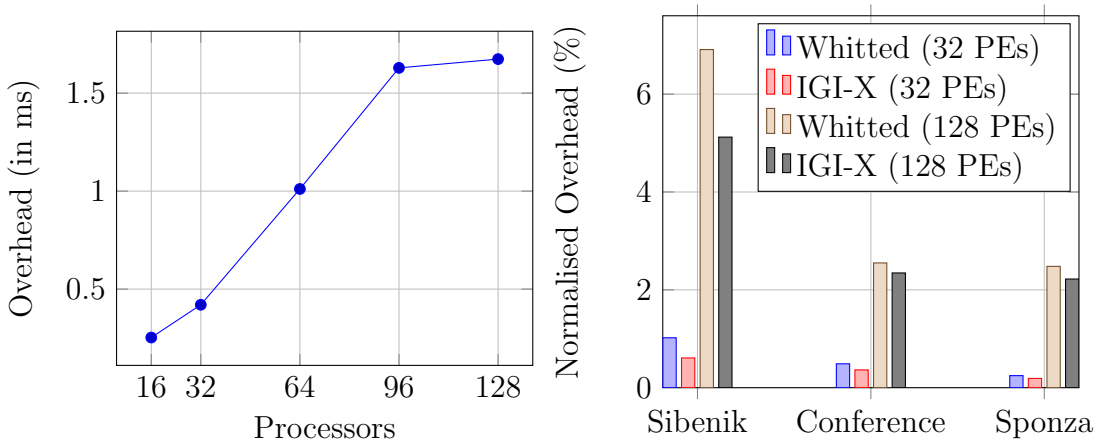


Figure 5.17: The effect of adding new jobs to the system on the performance existing jobs. J_0, J_1 and J_2 are started at intervals of $25s$ within each other. The resource partitioning is fixed, with the jobs being allocated 16, 32 and 64 workers respectively. The frame rate trend lines show that for each additional job added to the system, the existing ones are not impacted.

J_2 were created at 25 seconds from each other (J_0 at $0s$, J_1 at $25s$ and J_2 at $50s$). The configuration parameters for each of the three jobs can be seen in Table 5.2, under *Client Scalability*. The jobs are assigned 16, 32 and 64 dedicated workers respectively and neither job loses any resources during its execution. They are then monitored for approximately $200s$; Figure 5.17 shows the job performances in terms of frame rate. The frame rate of job J_0 fluctuates between 7.5 and 9 Hz. No discernible change in the performance of J_0 can be observed as job J_1 joins the system. The frame rate for job J_1 fluctuates between 7.5 and 8.5 Hz. The further addition of job J_2 appears to affect neither J_0 nor J_1, even though the former's resource allocation exceeds both combined. Moreover, J_2 also seems to settle into a stable frame rate of approximately 6 Hz. The trend lines for the frame rates of the three jobs are indicative that concurrent jobs, notwithstanding running on the same infrastructure, do not appear to affect each other. It should also be stressed that the rendering performance of the three jobs is not simply commensurate to the resource allocation but also tied to the complexity of the visualisation technique employed; this is the reason why notwithstanding an allocation of 64 workers, J_2 outputs at a slower frame rate than both other jobs.

Between the rendering of a frame and the next, jobs might gain (or lose) additional workers due to resource allocation and load balancing events. Lacking a priori knowledge of contributing workers, a synchronisation step is required to establish the actual participants and update state information. This overhead was quantified by recording the difference between wall-clock and actual frame generation time for a number of jobs. Figure 5.18a shows the actual overhead figures for the aggregated scenes on different worker configurations. The figure does not exceed 2 ms on average, even for a large number of workers. Moreover, the curve suggests that overhead is proportional to the number of workers employed, which is expected, since at the beginning of each frame, workers must synchronise with their Task Coordinator to receive changes in scene state information. Figure 5.18b expresses this overhead as a percentage of the frame rendering time. In particular, the scenes shown were rendered using two different techniques, IGI-X and Whitted-style, on two different worker configurations of 32 and 128 processors. In general, overhead is less than 2% of frame generation time, but for a large number of workers, fast rendering techniques may suffer penalties as high as 7%, since this overhead is dominated by communication costs.



(a) Average overhead. (b) Normalised overhead.

Figure 5.18: Average and normalised system overhead. The average overhead aggregates frame realisation penalties for different scene configurations and rendering techniques, while normalised overhead is given as a % of frame time segmented by scene, resource allocation and rendering algorithm.

5.6.3 Elasticity

The system needs to be elastic enough to respond quickly to user demands when the computation load increases and dispose of unutilised resources when it decreases. To measure the response of the system with respect to resource provisioning, two tests are carried out. In the first test, two jobs, J_0 and J_1, are submitted to the system and their worker allocations are varied using a sinusoidal function of time. The configuration parameters for the two jobs can be seen in Table 5.2, under *Elasticity I*. The results are shown in Figure 5.19; within approximately 300 seconds, the workers allocated to job J_0 scale from 48 up to 64 and down to 32 workers, which is reflected in the frame rate fluctuating between 8 and 12 Hz, while J_1's resources are scaled from 32 to 64 and back to 32, with the frame rate varying between 15 to 25 Hz. The number of workers W at time t are given by:

$$W = 48 + 16 \sin\left(\frac{2\pi(t+\phi)}{300}\right), \tag{5.7}$$

where $\phi = 0$ for J_0 and $\phi = 16$ for J_1. In the second test (see Figure 5.20), an equal-split resource partitioning scheme is assumed. Four jobs are admitted into the system with arrival times of $0, 100, 200$ and 400 seconds respectively. The configuration parameters for the four jobs is given in Table 5.2, under *Elasticity II*. J_0 starts with 128 workers; at time $t = 100$, J_1 is started, J_0's allocation is

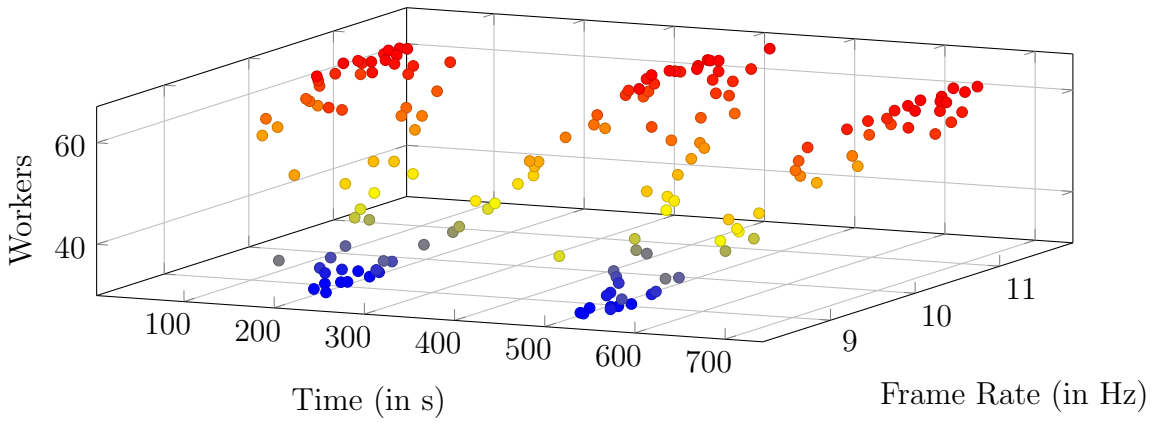
Experiment	Job	Scene	Technique	Arrival Time
Client Scalability	J_0	Šibenik	IGI-X	0
	J_1	Conference	IGI-X	25
	J_2	Sponza	PT	50
Elasticity I	J_0	Conference	IGI-X	0
	J_1	Šibenik	IGI-X	0
Elasticity II	J_0	Šibenik	Whitted	0
	J_1	Conference	Whitted	100
	J_2	Conference	IGI-X	200
	J_3	Šibenik	Whitted	400

Table 5.2: The table shows the job details for the scalability and elasticity tests. Rendering output was set to a resolution of 512 × 512 for all of the scenes listed.

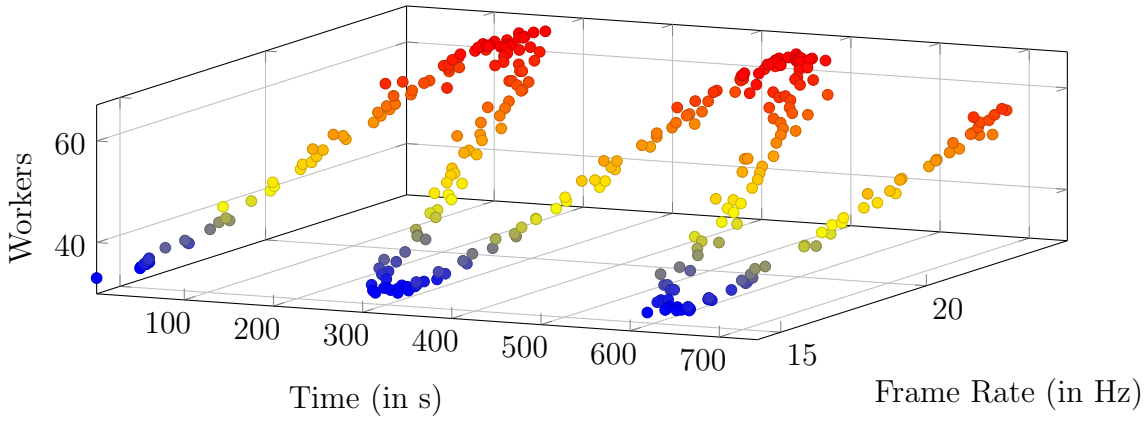
equally split between the two jobs, and so forth with the remaining J_2 and J_3. The system responds timely in both scenarios and is able to quickly bind and boot (though object lazy-loading, see §5.4.3) an idle resource to a running job, as well as promptly detach a resource from an active job and move it back to the free pool.

5.7 Discussion

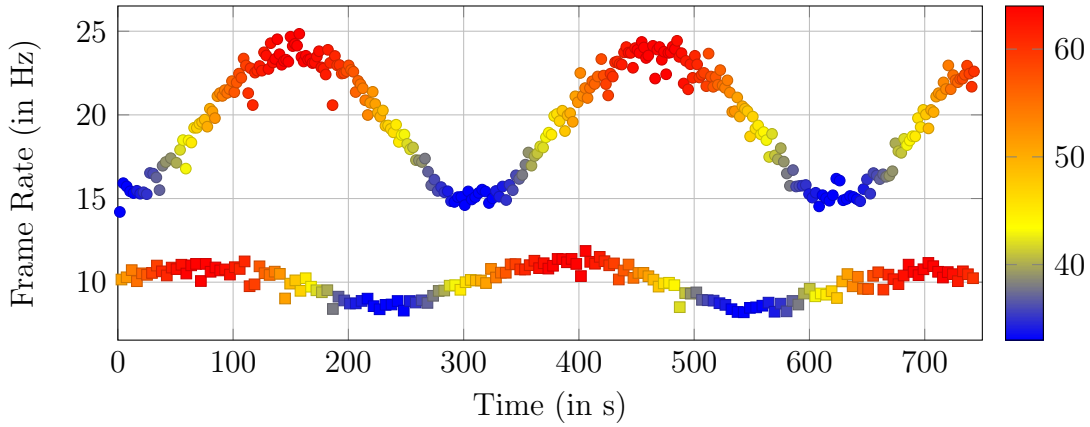
Figure 5.21 shows the parallel efficiency of the system for different resolutions and worker configurations. There is reasonable scalability for up to 64 processors, with an average parallel efficiency of 72% for both resolutions. Given the fixed problem size, it is expected for the parallel efficiency to decrease further as more processors are added. Reducing tile sizes to achieve a finer task granularity increases the relative communication and setup time, which become the dominant time factor. The overhead of synchronisation and the large bandwidth requirements per frame per second constrain the implementation to a maximum frame rate of 30 Hz. Rendering at high-definition (1080p), which nowadays is a common target, would reduce this figure significantly due to the higher bandwidth and computational requirements, especially for higher fidelity techniques. The rendering techniques have been implemented entirely on the CPU; in Figure 5.15b, it can be seen that compared with other works, the implementations are not well-optimised. For instance, Benthin *et al.* (2003) have recorded frame rates of up to 16 Hz for the Conference room rendered at video resolution (640 × 480), with 12 VPLs per



(a) Change in frame rate for job J_0 as resource allocation varies sinusoidally with respect to time.



(b) Change in frame rate for job J_1 as resource allocation varies sinusoidally with respect to time.



(c) Changes in frame rate for concurrent jobs J_0 and J_1 as resource allocation varies sinusoidally with respect to time.

Figure 5.19: The resource allocation for two concurrent jobs, J_0 and J_1, has been artificially throttled such that the number of workers assigned to each job varied sinusoidally against time, from 32 to 64. In 5.19a and 5.19b, the frame rate against the allocated number of resources for each job is shown. 5.19c shows the results for both jobs superimposed.

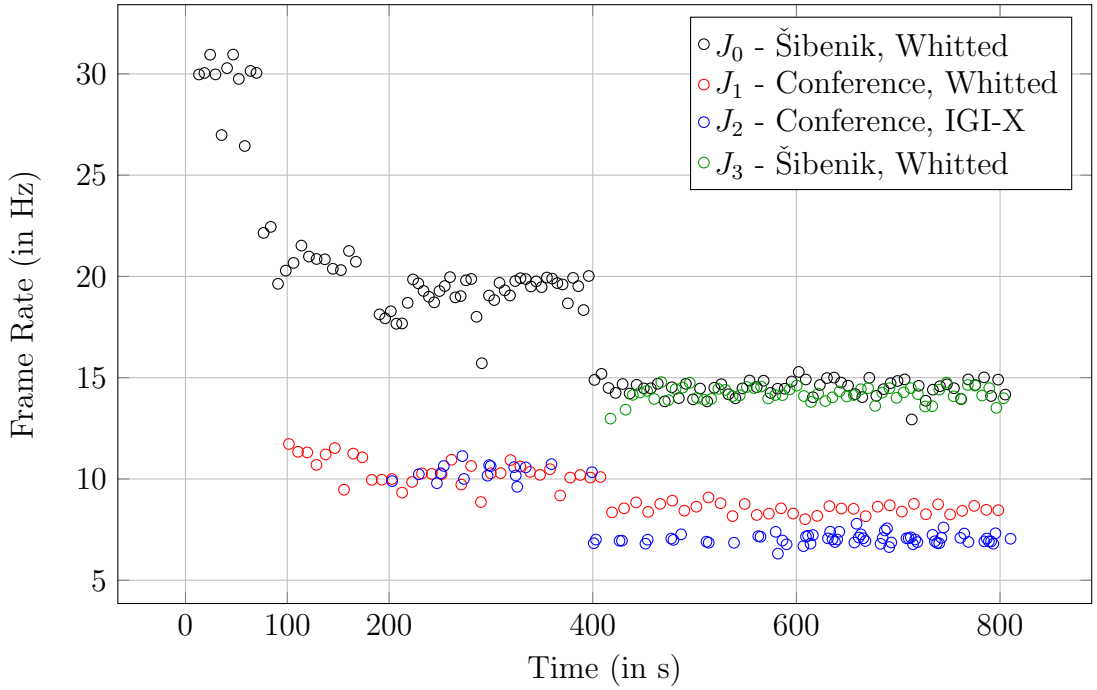


Figure 5.20: Effects of even resource distribution on frame rate.

pixel running on 48 cores, while our implementation of IGI runs at 3 Hz and IGI-X at 10 Hz, albeit both use 32 VPLs per pixel. In this case, the generality of the system works against it, because less assumptions can be made on the roles of workers. Benthin *et al.* (2003) use a well-defined system, where shared memory communication is employed between rendering threads running on the same physical machine. Within RaaS, the individual cores at any given node can be assigned to different jobs, and there is no guarantee that any assigned workers would be running on the same physical machine. Thus, all communication occurs via message passing.

The reprovisioning of resources in figures 5.19 and 5.20 show the system's capacity for elasticity. Particularly in Figure 5.19, at $145 \leq t \leq 155$, the system has to respond with a second to changes in the resource allocation of J_0. Thus, for the particular setup, a guarantee can be given that reprovisioning can occur within a second of a request being issued, provided physical resources are available. Resource migration, which determines how resources are detached from one job and assigned to another, was targeted at interactive systems, and thus, resources are detached at specific checkpoint during synchronisation stage. Hence, once a frame starts being computed, a resource cannot be preempted into idle state. Jobs running with very high-quality settings may degrade into non-interactivity

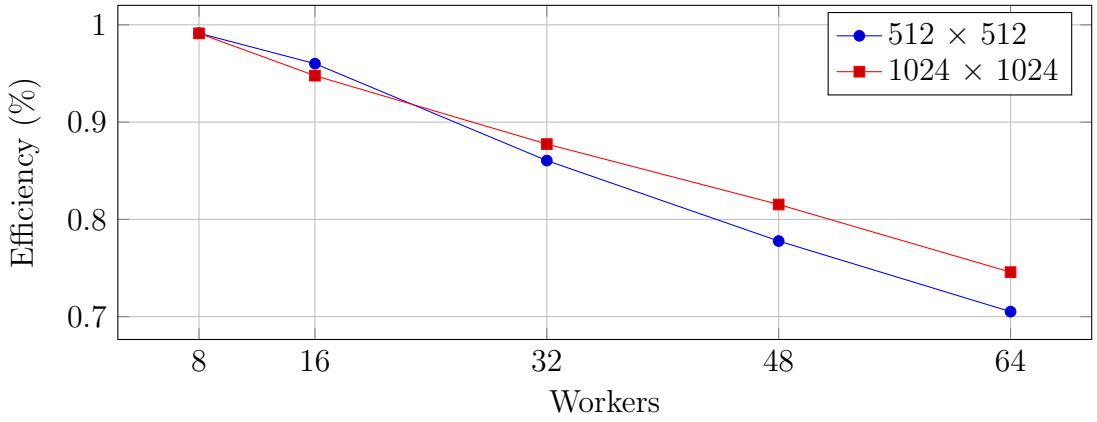


Figure 5.21: Parallel efficiency for different worker configurations.

and increase the time a resource takes to switch to idle. Resources revert to idle state whenever they have been released, even when explicitly released for the purpose of allocation to another job. The release-request cycle is not atomic; thus, a resource released to be allocated to a job, say J_1, may be out run by another job J_2, which issues its request, after the release of the resource, but prior to its allocation to J_1. This behaviour is triggered by the way release-request mechanism works: when a job requires additional resources, a request is made to the Resource Manager. If no available resources are found, a release command may be sent to one or more jobs, to force them to relinquish some resources. The original requester may then try to ask for the resources again. To prevent a job running ahead of another during an allocation, unsatisfied requests could be queued, with the requesting job being informed once the resources are available, via a callback mechanism. This solution could also solve any potential starvation problems that are introduced due to the non-atomic nature of resource allocation.

This work was carried out during a period where GPUs were becoming more relevant in the acceleration of high-fidelity rendering; in hindsight, the use of GPUs could have helped reduce computation times and consequently, the number of workers, in turn leading to greater possibilities of batching during both processing and communication. This is especially true in the light of the advances in virtualisation of GPUs and the adoption of GPU cloud computing by services vendors. Furthermore, it is clear that a cost model for the usage of various cloud resources would have given insight on the financial advantages of adopting a cloud service for rendering as opposed to constructing dedicated rendering clusters, if any. Although this fact is acknowledged, the focus of this work was the compu-

tational aspect of rendering as a service rather, specifically the scalability and elasticity of such a system.

5.8 Summary

This chapter presented RaaS, a novel system for rendering as a service and a first attempt towards a scalable and elastic solution for interactive high-fidelity rendering in the cloud. Multiple reference high-fidelity rendering techniques have been used to measure the performance of the system. Progressive rendering and temporal filtering are used to reduce discontinuities between different levels of solution convergence. The system employs object lazy-loading to ensure fast and efficient resource binding, required for an elastic solution.

RaaS provides a many-to-many connectivity pattern between clients and server resources. It also furnishes resources with rapid elasticity to allow for almost instantaneous de-provisioning and reprovisioning to adapt to workload changes. Please refer to Table 9.3 for details.

The results show that even though RaaS addresses but some of the issues that may arise in the realisation of such a system in something as amorphous as the cloud, it is nonetheless a promising first step in the direction of a system for economically providing a step change in the fidelity of rendering solutions achievable to most clients. While the current implementation does not achieve the highest of frame rates it serves to demonstrate the potential of such a method and such levels of interactivity may still be used by the engineer and artist for quasi real-time performance.

CHAPTER 6

Remote Asynchronous Indirect Lighting (RAIL)

Modelling of global illumination increases the level of realism and immersion in virtual environments. While a large number of methods for computing graphics of higher fidelity have been developed, they typically trade off quality for performance and are incapable of running on all but the machines with the highest specifications. This chapter proposes Remote Asynchronous Indirect Lighting (RAIL), an extension to the rendering as a service paradigm, described in Chapter 5, that decouples inexpensive computations, such as the synthesis of the direct lighting component, from the rest of the rendering pipeline, and moves their execution to the client device. The decoupling of different lighting components allows the system to provide an asynchronous computation method that is suitable for low-latency, high-resolution rendering prevalent in simulations, computer games and other real-time applications that require high-fidelity interactive graphics, making it overall faster than the work presented in Chapter 5.

The chapter is structured as follows: §6.1 introduces the chapter and outlines the respective contributions, §6.2 gives overview of RAIL as a centralised global illumination algorithm, §6.3 discusses the decomposition of RAIL into a distributed rendering pipeline, §6.4 and §6.5 demonstrate results from RAIL followed by a discussion and §6.6 concludes the chapter.

6.1 Introduction

RAIL has minimal bandwidth and client-side computation requirements. It can achieve client-side updates at 60 Hz or more, at HD and UHD resolutions. A

component-based approach is used in the computation of the global illumination solution, which is split into direct and diffuse indirect lighting components. The indirect lighting is decoupled from the rest of the rendering and computed remotely due to its computational expense. Different approaches are used for static and dynamic geometry. In the latter case, a higher-order ambient function is used to coarsely approximate indirect lighting. This service is provided to clients of multi-user environments, thus amortising the cost of computation over all connected consumers of the service. Computation results are stored in an efficient object space representation that does not require the large bandwidths seen in previous work. Another advantage over related work is the resolution agnostic representation of indirect lighting, where increasing client display resolution does not increase bandwidth requirements. The reconstruction of indirect lighting on the client is lightweight and efficient, making this method suitable for any device that supports basic hardware acceleration functionality. The method scales well to many clients connected to the same service, achieving a significant overall boost in performance via amortisation of computation. The contributions of this chapter are:

- a novel fast algorithm for multi-bounce global illumination based on instant radiosity methods
- a scalable asynchronous distributed rendering algorithm with low bandwidth requirements that is resolution-independent and robust to network service fluctuations
- a novel higher order ambient function for approximating diffuse indirect lighting
- the application of RAIL to RaaS for highly interactive, high-fidelity rendering in HD (and higher), over a wide spectrum of devices

6.2 Method

The rendering method proposed in this work is described by the following sequence of steps:

1. Generate point cloud for sampling indirect diffuse lighting (offline precomputation) (§6.2.1)
2. Use light tracing to generate VPLs (§6.2.2)
3. Progressively shade point cloud representation using VPLs (§6.2.2)
4. Reconstruct observer's view from shaded point cloud (§6.2.2)
    - Static geometry is shaded using interpolated irradiance from the point-cloud (§6.2.3)
    - Dynamic geometry is shaded using a higher-order ambient function, a coarse approximation of indirect lighting (§6.2.4)
5. Combine reconstructed view with direct lighting and present (§6.2.3)

Section 6.2 describes a centralised and synchronous version of the proposed rendering algorithm, elaborating on how a point description of the scene is generated and used to sample the diffuse indirect lighting function on object surfaces and its updating and reconstruction in real-time from these sparse samples. Subsequently, in Section 6.3 the asynchronous version of the algorithm is introduced, together with its distribution to a client-server setting which is amenable to the rendering as a service paradigm presented in Chapter 5.

6.2.1 Selecting Indirect Diffuse Sample Points

Ward *et al.* (1988) proposed to sparsely sample the indirect lighting function in such a way as to minimise error during reconstruction, while Keller (1997) proposed the use of virtual point lights to estimate diffuse indirect lighting at a point on a surface; Debattista *et al.* (2009) combine ideas from these works into the instant caching method (see §3.5.4), to accelerate the computation of indirect lighting for real-time rendering, notably for dynamic scenes. Both Debattista *et al.* (2009) and Ward *et al.* (1988) use rays from the observer to trigger on-demand sampling of the indirect contribution, storing the results in an acceleration structure, an octree. To generate samples covering the entire scene using

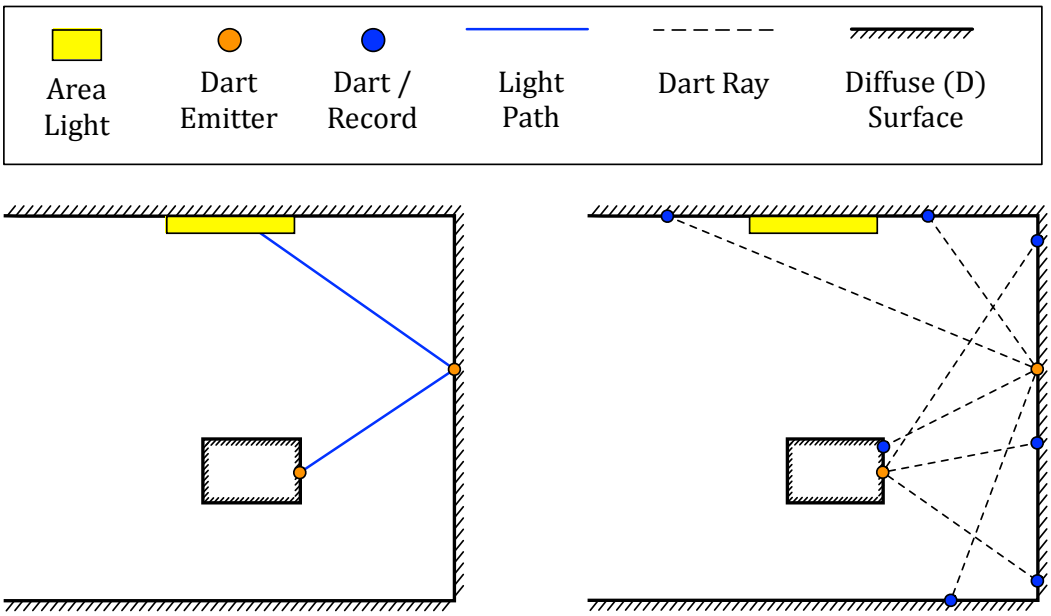


Figure 6.1: Light tracing coupled with dart-throwing to generate point cloud scene representation. The light tracing process, shown on the left, is used to find suitable surface points at which to deposit the dart-throwing sources. In a second step, shown right, the sources throw a number of darts and record their points of intersection, which are then used as sample points into the indirect lighting function on the respective surfaces.

such view-based methods, multiple cameras must be employed. A straightforward alternative could be that of converting triangle faces into a discrete set of surface points, although such a method does not discriminate against surfaces that are unreachable by any light paths and can potentially generate larger and less spatially efficient point sets. Brouillat *et al.* (2008) combine photon mapping (Jensen, 2001) with irradiance caching to provide scene-wide coverage of indirect lighting records without using multiple camera views, while also avoiding user intervention. The selection of indirect diffuse sample points in this chapter is based on the work by Bikker & Reijerse (2009), who use a method similar to Brouillat *et al.* (2008) where light tracing is used to place a number of dart sources in the scene (Figure 6.1, left). A dart throwing technique is then applied, using these sources to trace and record points on the surface of scene geometry (Figure 6.1, right). The dart density is controlled via poisson disk sampling, to ensure that an area is neither too densely nor too sparsely populated. The area of effect of a dart, or record, is determined by evaluating the ambient occlusion function at the respective point in the scene (Langer & Bülthoff, 1999). This is illustrated

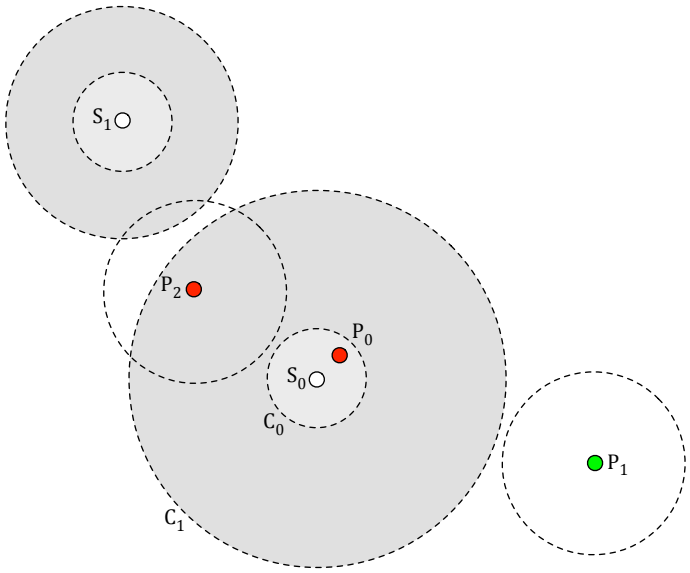


Figure 6.2: Point generation using Poisson disk sampling. The red samples, P_0 and P_2 denote rejected samples, while the green sample, P_1, has been accepted. P_0 is rejected using an *early-out* strategy; it is immediately discarded because its position falls within the inner circle C_0 of S_0, which is the minimum distance between two accepted samples. On the other hand, P_2 has been rejected after its area of effect had been calculated and found to intersect S_0's.

in Figure 6.2, where the candidate records, P_0, P_1 and P_2, are tested against the already computed records, S_0 and S_1. If a record lies within some minimum distance from a valid sample, illustrated by the inner circle C_0, it is discarded immediately (e.g. P_0). The outer circle, C_1, is the area of effect of record S_0, and is a function of the ambient occlusion at S_0. Candidate samples P_1 and P_2, which fall outside of C_0, have their ambient occlusion term evaluated and the respective outer circles tested for intersection with C_1. P_2 tests positive and is thus rejected, while P_1 is accepted. Figure 6.3 shows examples of the generated point sets.

6.2.2 Estimation and Reconstruction of Indirect Lighting

The selection of a point cloud representation of the scene for sampling the indirect lighting function is a one-time, per-scene precomputation step. The estimation of indirect lighting, however, is a dynamic process that is affected by scene changes in illumination sources or objects. This entails evaluating the rendering equation (Kajiya, 1986) over the hemisphere centred at each sample point. To accomplish

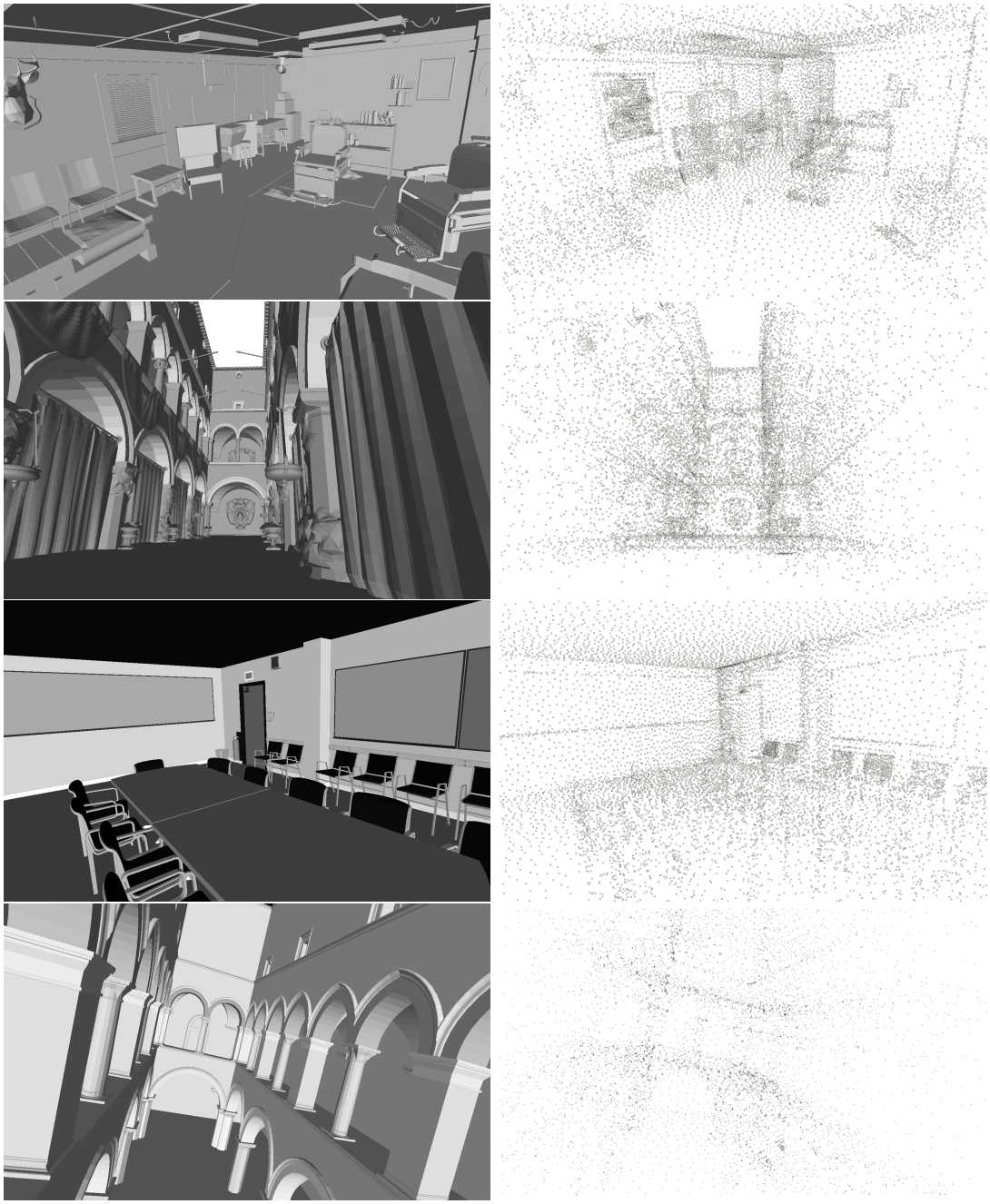


Figure 6.3: Parallel view of geometry (left) and generated point sets (right) for the scenes used in this chapter.

this, we use a many-lights algorithm (Dachsbacher *et al.*, 2014); specifically, we compute the indirect lighting contribution in two phases, first by tracing a number of light paths and creating VPLs for each path vertex, and secondly by iterating over the VPLs and computing the respective contribution for each unoccluded sample point.

The indirect lighting contribution at a point is progressively accumulated when no changes in the scene state are recorded. Specifically, for n progressive contributions, the indirect lighting at the sample point is the mean of these contributions $E = (\sum_{i=1}^{n} E_i) \,/\, n$, where E_i is the i^{th} contribution and E the current indirect lighting at an arbitrary sample point. When a change in scene state is effected, a new sequence of contributions is started, to match changes in state. However, to avoid abrupt changes in the lighting, the first term of the new sequence is carried over from the previous sequence ($E_0 = E$). Although this introduces additional bias, it also favours temporal coherence and shows less discontinuity.

The estimation step updates a point cloud Q such that each sample point $\mathbf{q} \in Q$ represents a point on a surface at which diffuse indirect lighting has been evaluated. To reconstruct the indirect lighting function for a point $\mathbf{p}$ on any surface in the scene, a form of inverse distance weighting is used to interpolate the irregularly-spaced data in Q (Shepard, 1968). The weight of the contribution of each sample $\mathbf{q}$ in the extrapolation of $\mathbf{p}$ is given by:

$$W(\mathbf{p}, \mathbf{q}) = (\max(0, r - |\mathbf{p}_p - \mathbf{q}_p|) \max(0, \mathbf{p}_n \cdot \mathbf{q}_n))^{\mu}_{,} \tag{6.1}$$

where r is the range determining $\mathbf{q}$'s area of effect and, $\mathbf{p}_p$, $\mathbf{q}_p$ and $\mathbf{p}_n$, $\mathbf{q}_n$ are the positions and surface normals at $\mathbf{p}$ and $\mathbf{q}$ respectively. The exponent μ determines whether the reconstruction function Φ peaks ($0 < \mu \leq 1$) or is level ($\mu > 1$) at the nodes (Pál *et al.*, 2009):

$$\Phi(\mathbf{p}) = \frac{\sum_{\mathbf{q} \in Q} W(\mathbf{p}, \mathbf{q})\, \mathbf{q}_e}{\sum_{\mathbf{q} \in Q} W(\mathbf{p}, q)}. \tag{6.2}$$

Q is the set of all indirect samples (the point cloud containing the estimated irradiance) and $\mathbf{q}_e$ is the irradiance at $\mathbf{q}$.

6.2.3 Integrating Direct and Indirect Lighting

Reconstructing the indirect lighting contribution requires knowledge of point cloud Q and geometric details about the points for which the function is being reconstructed, such as surface positions, normals and albedos. In deferred shading pipelines, which perform screen-space shading, this information is readily available as a geometry-buffer (G-buffer) (Deering *et al.*, 1988; Saito & Takahashi,

1990). This makes reconstruction using Equation 6.2 straightforward but inefficient, since all points in Q have to be considered, even though they might not contribute anything to the final value. Thus, a multiple-reference regular grid is used to partition and store a representation of Q and accelerate nearest neighbour queries using spatial hashing. Records are referenced from each cell that overlaps their sphere of validity, simplifying the lookup to the examination of a single cell and the records contained within.

From Equations 6.1 and 6.2, it follows that in order to calculate the contribution of each sample $\mathbf{q}$, properties like position, surface normal, irradiance and range are required and thus, have to be accounted for in the space complexity of the regular grid. Specifically, let c be the number of cells along an edge of the grid and c^3 the total number of cells in the grid. The edge of each cell is c_l units long. The sample range of effect r in the weighting function (Equation 6.1) determines the spatial extent of a search operation and can be used to estimate the maximum number of references for a sample to $(2r \,/\, c_l)^3$, with the total being $|Q| \cdot (2r \,/\, c_l)^3$. Each cell holds an index to a record reference, which points to the first record in a bin that holds references to records affecting the given cell. A further indirection exists, that maps each reference to the actual record index, and finally the records themselves are stored. The index held in each cell of the grid is 4 bytes long, while each entry in the reference map is 8 bytes long. A single record is 23 bytes long, 12 bytes for position, 8 for surface normals and 3 for irradiance. The upper bound on the space requirements of the regular grid is thus $|Q| \cdot (8(2r \,/\, c_l)^3 + 23) + 4c^3$. A typical grid ($c = 64, c_l = 4, r = 6$ and $|Q| = 40000$) requires 10 MB of GPU memory. The records are stored in memory as a structure-of-arrays (SoA) rather than an array-of-structures (AoS), the motivation being that only irradiance values in records change and thus locality during host-to-adapter copies is exploited by modifying in bulk only the affected array. Thus, using a G-buffer and a regular grid storing irradiance samples, the diffuse indirect lighting for the current view can be efficiently reconstructed. High-frequency texture details are not present in the reconstruction and must be added via multiplicative blending of the albedo G-buffer channel and the indirect lighting. Subsequently, the result is blended with the direct lighting output of the rendering pipeline. For fill-limited devices, the reconstructed irradiance channel may be smaller than the size of the frame buffer, to preserve high frame rate interactivity. In these cases, prior to the final composition, geometry-aware upscaling using joint bilateral upsampling (Kopf *et al.*, 2007) is applied to the

channel. Similarly to McGuire & Luebke (2009), the weights used are based on 2D bilinear interpolation, normals and depth differences between the low and high resolution G-buffers. The upscaled irradiance is combined with the albedo channel and then with the direct contribution, to derive the final image (Figure 6.4).

6.2.4 Dynamic Scenes

The system currently supports dynamic objects, deformable ones also. These objects are factored in the VPL tracing and point cloud shading steps (§6.2.2), and thus, can occlude and reflect light. To avoid changing the structure of the illumination grid (§6.2.3) whenever an object moves, a coarser approximation of indirect lighting is used, inspired by ambient occlusion, that is computed in two steps. The constant ambient term, traditionally used to approximate indirect reflections in the scene, is turned into a higher-order function of space; this is then evaluated by partitioning the scene into a coarse regular grid, with each cell containing an ambient term approximation for the region (see §6.2.5), and trilinearly interpolating these values across adjacent cells (see §6.2.6). The ambient term for each cell is computed from the weighted mean of all irradiance points in Q affecting the cell. This coarse grid is referred to as the *ambient grid*.

6.2.5 Building the Ambient Grid

The ambient grid A represents a coarse approximation of diffuse indirect lighting in the scene and is generated from the point set Q. It is distinct from the grid structure discussed in §6.2.3, which is used to accelerate nearest neighbour searches of Q. Spatially, A is fitted over the scene geometry such that the longest edge of the bounding volume of the scene corresponds to an edge of the bounding volume of A. The edges of the ambient grid are equal which means that geometrically, it is a cube, subdivided equally along all edges. The constituting cells contain three important values, a single quantity A_e representing the weighted mean irradiance (*ambient contribution*) of the sample points in Q which are also contained with the volume of the cell itself, a normalised vector in $A_n \in \mathbb{R}^3$, which represents the principal direction of the sample normals contained within the cell, and a scalar contribution count A_c that keeps track of the number of ambient contributions a given cell has received from neighbouring cells. Cells

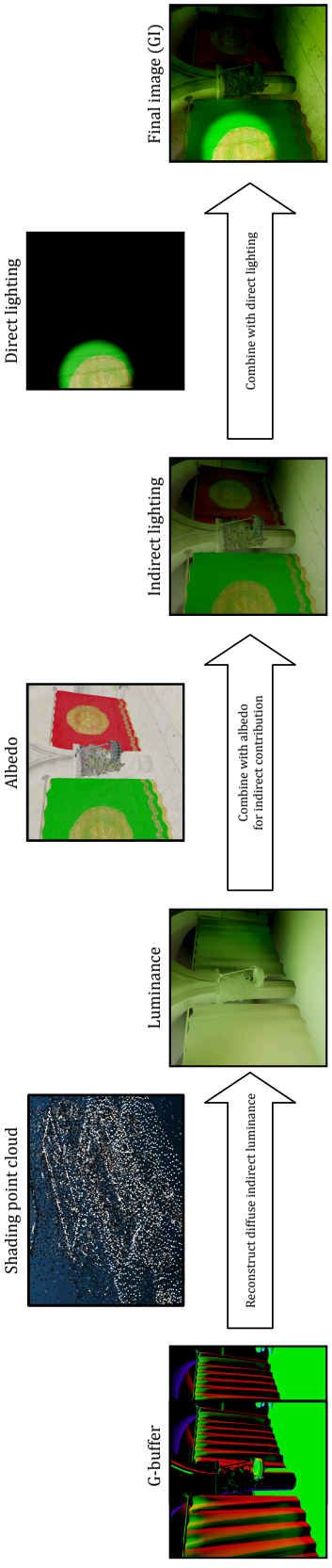


Figure 6.4: The client rendering pipeline reconstructs indirect lighting from the shading point cloud and G-buffer, merging the result with direct lighting to obtain a GI solution.

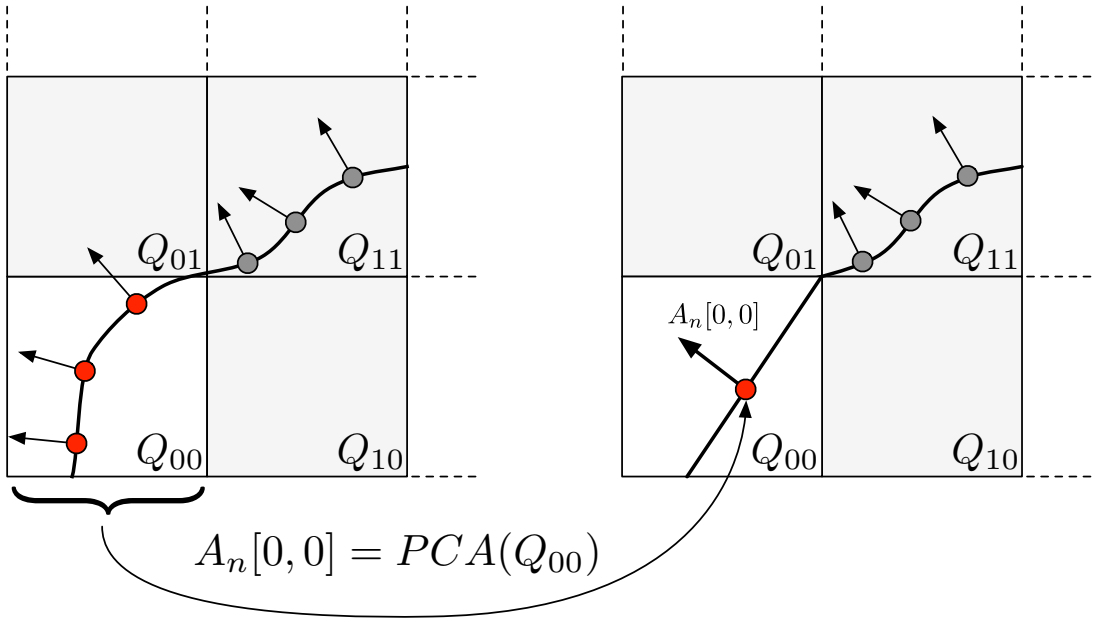


Figure 6.5: Principal component analysis is performed on the samples in cell Q_{00} to determine surface normal $A_n[0,0]$.

are indexed via a triple (x, y, z), where $0 \leq x, y, z < subdivisions$. The ambient contribution for a cell is computed as follows:

$$A_e[x, y, z] = \frac{\sum_{\mathbf{q} \in Q_{xyz}} \mathbf{q}_o \, \mathbf{q}_e}{\sum_{q \in Q_{xyz}} \mathbf{q}_o}, \tag{6.3}$$

where Q_{xyz} is the subset of points in Q that is contained in the grid cell with index (x, y, z), and $\mathbf{q}_e$ and $\mathbf{q}_o$ are the irradiance and ambient occlusion values for sample $\mathbf{q}$ respectively. Note that $\mathbf{q}_o$ is computed offline, during the generation of Q, and stored (see 6.2.1). The layout of scene geometry, and consequently the distribution of Q, may be such that some cells in the grid are empty (i.e., $Q_{xyz} = \emptyset$). A straightforward iterative approach is used to propagate the ambient contribution from neighbouring cells and fill empty ones. When the ambient grid is first created, principal component analysis is performed on the normal vectors of the samples contained in each cell. A principal component is determined and used as the aggregated surface normal A_n for that cell; this is illustrated in Figure 6.5 using a two-dimensional grid.

Propagation is a runtime process which uses the ambient contribution values of neighbouring cells to populate the empty ones; the process is described in Algorithm 7. For each cell in the grid that has a valid ambient contribution (*source cell*), the direction of propagation is determined from the principal normal A_n.

The normal is used to determine which neighbouring cells to consider. The selection strategy is illustrated in Figure 6.6. The principal normal $\mathbf{n} = A_p[p_x, p_y, p_z]$ is decomposed into its axial components n_x, n_y and n_z. If the length of each component exceeds a given threshold ($\pm\frac{1}{\sqrt{3}}$), then the cells adjacent to the face with the component axis' normal are added to the propagation set. Consequently, the ambient contribution of the source cell is combined with that at each of the selected neighbours; A_c keeps track of the number of contributions a cell has received during one iteration of propagation. An iteration completes once all the source cells have been considered, after which the propagated contributions are averaged. Figure 6.8 gives an example of propagation, for two iterations. The threshold value $\frac{1}{\sqrt{3}}$ is the length of each component of a unit vector which makes an angle of $\pi/4$ to each principal axis.

6.2.6 Using the Ambient Grid

During rendering, the ambient grid is used to contribute indirect lighting to regions that are not covered by the samples in Q, such as dynamic geometry. Irradiance at $\mathbf{p}$ is coarsely estimated from the ambient contributions by performing trilinear interpolation between adjacent cells of the grid. Let $f_l(x, y, t)$ linearly interpolate between x and y, for $t \in [0, 1]$, such that:

$$f_l(\mathbf{a}, \mathbf{b}, t) = \mathbf{a}(1 - t) + t\mathbf{b}. \tag{6.4}$$

A bilinear function, f_b, can be constructed from the product of two linear interpolations (see Figure 6.7), such that:

$$f_b(\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{t}) = f_l\left(f_l(\mathbf{a}_0, \mathbf{a}_1, t_x), f_l(\mathbf{a}_2, \mathbf{a}_3, t_x), t_y\right). \tag{6.5}$$

Subsequently, a trilinear interpolation function f_t can be constructed from the linear interpolation of the output of two bilinear interpolation functions (see Figure 6.7):

$$f_t(\mathbf{a}_0, \cdots, \mathbf{a}_7, \mathbf{t}) = f_l\left(f_b(\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{t}), f_b(\mathbf{a}_4, \mathbf{a}_5, \mathbf{a}_6, \mathbf{a}_7, \mathbf{t}), t_z\right). \tag{6.6}$$

Algorithm 7 Propagation of ambient contribution values to empty cells.

```
1: procedure PROPAGATE(grid, iterations)
2:     A_c[···] ← 1
3:     for 1 ≤ n ≤ iterations do
4:         for each p ∈ grid ∧ ¬ isEmpty(p) do
5:             cells ← FilterNeighbours(p)
6:             for each c ∈ cells∧ isEmpty(c) do
7:                 A_e[c_x, c_y, c_z] ← A_e[c_x, c_y, c_z] + A_e[p_x, p_y, p_z]
8:                 inc A_c[c_x, c_y, c_z]
9:             end for
10:        end for
11:        for each p ∈ grid do
12:            A_e[p_x, p_y, p_z] ← A_e[p_x, p_y, p_z]/A_c[p_x, p_y, p_z]
13:        end for
14:    end for
15: end procedure
16: procedure FILTERNEIGHBOURS(p)
17:    n ← A_n[p_x, p_y, p_z]
18:    cells ← ∅
19:    if n_x >= 1/√3 then
20:        cells ← cells ∪ rightNeighbours(p)
21:    else
22:        if n_x <= −1/√3 then
23:            cells ← cells ∪ leftNeighbours(p)
24:        end if
25:    end if
26:    if n_y >= 1/√3 then
27:        cells ← cells ∪ topNeighbours(p)
28:    else
29:        if n_y <= −1/√3 then
30:            cells ← cells ∪ bottomNeighbours(p)
31:        end if
32:    end if
33:    if n_z >= 1/√3 then
34:        cells ← cells ∪ frontNeighbours(p)
35:    else
36:        if n_z <= −1/√3 then
37:            cells ← cells ∪ backNeighbours(p)
38:        end if
39:    end ifreturn cells
40: end procedure
```

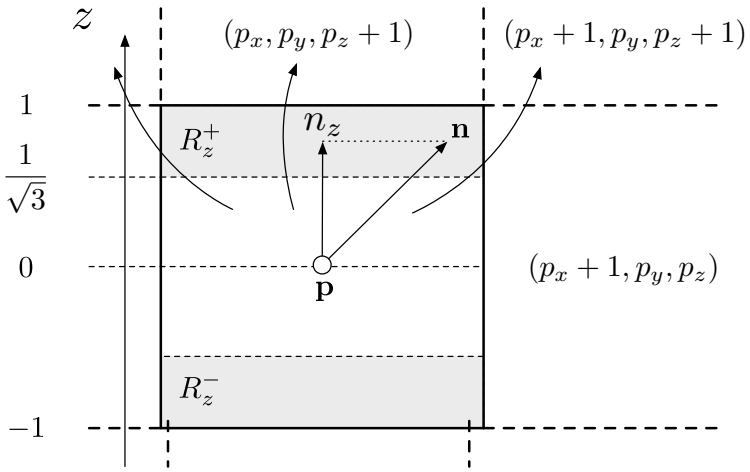


Figure 6.6: Selection of neighbouring cells for propagation. The cells indicated by the arrows are added to the propagation set *cells* when the normal vector n falls within the designated region R_z^+, that is, $n_z \geq \sqrt{1/3}$.

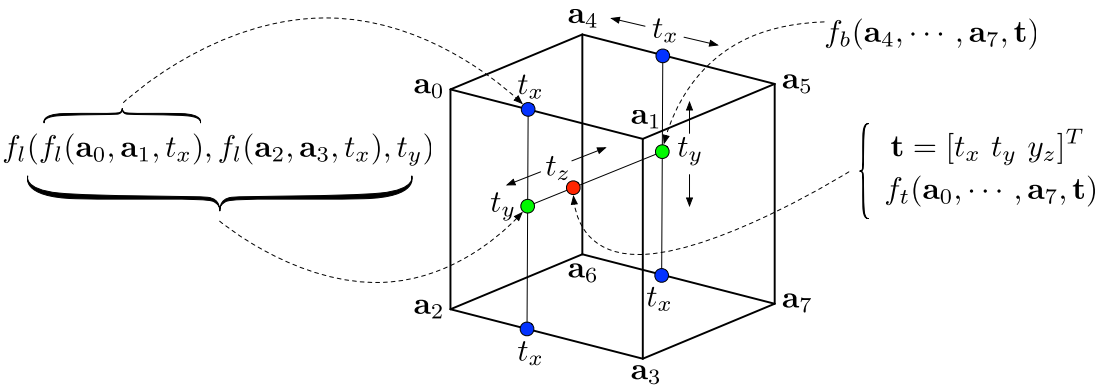


Figure 6.7: Trilinear filtering is used to interpolate between the ambient component values stored at adjacent cells in the ambient grid A. The function f_t is a composition of linear interpolation functions f_l. For convenience, f_b is defined as the composition of two linear interpolation functions, such that $f_b(\cdots) = f_l(f_l(\cdots), f_l(\cdots), \cdots)$, and subsequently, f_t is defined in terms of both linear and bilinear interpolation functions. The interpolation space is assumed to be a unit cube.

(a) No ambient value. (b) 1^{st} iteration. (c) 2^{nd} iteration.

Figure 6.8: Iterative propagation process for ambient lighting. In 6.8a, cells that contain no samples (and hence no ambient value) are marked in white. $A_e[i,j]$ marks the weighted mean of irradiance values, which is used to propagate indirect lighting to adjacent cells. In 6.8b, one iteration of propagation has been performed, and the empty cells adjacent to (i,j) which face the hemisphere of directions around $A_n[i,j]$ have received indirect lighting. Particularly, $A_n[i,j]$ acts as an occluder of sorts, to stop propagating values in its opposite direction. Cells that do not have a normal vector defined due to being empty of irradiance samples propagate values in all directions, that is, to all empty adjacent cells. 6.8c shows the ambient grid after two iterations of propagation, where all the empty cells are now populated with an ambient value.

Thus, the higher-order ambient function, $F_a(A, \mathbf{p})$, is given by:

$$\begin{aligned} F_a(A, \mathbf{p}) = f_t(&A_e[r_x, r_y, r_z], A_e[r_x + 1, r_y, r_z], A_e[r_x, r_y + 1, r_z], \\ &A_e[r_x + 1, r_y + 1, r_z], A_e[r_x, r_y, r_z + 1], A_e[r_x + 1, r_y, r_z + 1], \\ &A_e[r_x, r_y + 1, r_z + 1], A_e[r_x + 1, r_y + 1, r_z + 1], \mathbf{s}), \end{aligned} \tag{6.7}$$

where $\mathbf{r} = int(\mathbf{p})$, which returns the integer part of the components of $\mathbf{p}$, such that (r_x, r_y, r_z) is the index of the grid cell containing point $\mathbf{p}$; $\mathbf{s} = frac(\mathbf{p})$, which returns the fractional components of $\mathbf{p}$, such that $s_x, s_y, s_z \in [0, 1)$. The application of F_a to an object subject only to direct lighting can be seen in figures 6.9a and 6.9b. In the first figure, the object is illuminated only by means of direct light; since the light is partially occluded, a great part of the object is depicted in black. In the second figure the black patches on the object are now lit via the ambient function, albeit the latter does not take into account surface occlusion. Thus, a heavily occluded point receives the same contribution as one that is less occluded, leading to a loss of perception of the shape of the object, as can be observed in Figure 6.9b. In order to curtail a point's exposure to ambient lighting and provide a better perception of the shape of geometry (Langer & Bülthoff, 1999), ambient occlusion (AO) is applied to F_a. The extension to the ambient function is thus:

$$F_{ao}(A, \mathbf{p}) = \frac{F_a(A, \mathbf{p})}{\pi} \int_{\Omega} V(\mathbf{p}, \omega)\ \omega \cdot N_p\ \mathrm{d}\omega. \tag{6.8}$$

AO being a global method, it requires access to scene geometry in order to compute point visibility, which makes it less suitable for use in rasterisation. Instead of traditional AO, screen space ambient occlusion (SSAO) is used, which is a faster approximation that computes occlusion from neighbouring pixel depths rather than scene geometry (Mittring, 2007). The application of the new ambient function F_{ao} can be seen in 6.9c, where the AO term is evaluated using SSAO. Figure 6.10 shows how a typical dynamic object affects and is in turn affected by indirect lighting during rendering; the scene is entirely lit using the spatially-varying ambient function.

(a) Direct lighting

(b) 1^{st} Step: Trilinear interpolation across ambient terms of adjacent cells using uses F_a.

(c) 2^{nd} Step: Ambient occlusion using F_{ao}.

Figure 6.9: Indirect lighting function for dynamic objects, where 6.9a is the base direct lighting, 6.9b shows the object shaded with the spatially-varying ambient term, and 6.9c augments the result of 6.9b with screen space ambient occlusion.

Figure 6.10: Dynamic object contributing to and receiving diffuse indirect lighting. The entire scene has been visualised using the coarse grid described in §6.2.4 and does not use irradiance information contained in point cloud Q.

6.3 Distributing the Rendering Pipeline

In this section, the synchronous rendering method described thus far is transformed into an asynchronous distributed rendering pipeline; the method is synthesised into a number of steps in §6.2 and also shown in Algorithm 8. The first three steps (lines 1-3) are computationally expensive, beyond the reach of low-end hardware such as smartphones and tablet devices, making them good candidates for offloading to a powerful server backend. Particularly, the first step (line 1), which generates the point set, is a one-time precomputation step that can be carried out offline. The last two steps (lines 4-6) may be easily run on a low-end device, provided Q is available, in the form of the regular grid (§6.2.3). Distributing the rendering pipeline, thus, becomes a problem of synchronisation, whereby the regular grids at the clients' end are made consistent with that at the server, and the server's representation of dynamic objects in the scene, such as light sources, is made consistent with that of its clients.

Algorithm 8 Synchronous version of the proposed rendering algorithm.

1: $Q \leftarrow$ GenerateSamplingPointSet(*scene*)
2: $V \leftarrow$ TraceVPLs(*scene*)
3: ShadeSamplingPointSet(Q, V)
4: $I \leftarrow$ ReconstructIndirectLighting(*scene*, Q)
5: $D \leftarrow$ ComputeDirectLighting(*scene*)
6: MergeAndPresent(I, D)

The highly interactive nature of client applications precludes the use of a blocking synchronisation mechanism that depends on network performance. Thus, a particular grid G_c at a client device is treated as a local cache of the server version G_s, and is updated asynchronously, without affecting the local rendering steps (lines 4-6), which are allowed to run unconstrained. Server-side, G_s is updated to reflect indirect diffuse lighting in the scene (2-3) and executes independently of client communication. Any scene changes received from clients are queued and applied to scene state on the server, invalidating indirect lighting computed thus far. The interplay of these components is illustrated in Figure 6.11.

6.3.1 Synchronisation of Indirect Lighting

The first message exchange between a client and the server backend is the initial transfer of grid G_s to the client, such that $G_c = G_s$. Subsequent exchanges are always started by the client device, which sends camera details and any changes to scene state that potentially affect indirect lighting, such as changes in light sources. In response, the server uses the client's camera parameters to form a frustum and clip the point cloud. The points contained in the frustum are then sent back to the client, together with any changes performed to scene state by other clients.

Frustum clipping operates at cell-based granularity (see Figure 6.12). Thus, all points contained in a cell intersecting the frustum are included in the response message, provided the irradiance values of at least one of them has changed. Particularly, the structure of a response contains a unique identifier for the cell, followed by the irradiance triples for the points contained in that cell. Since the point ordering is deterministic, indices are not included with the irradiance values; a map at the client associates the position of each irradiance value for a received cell to a position in the SoA. This significantly reduces the size of data

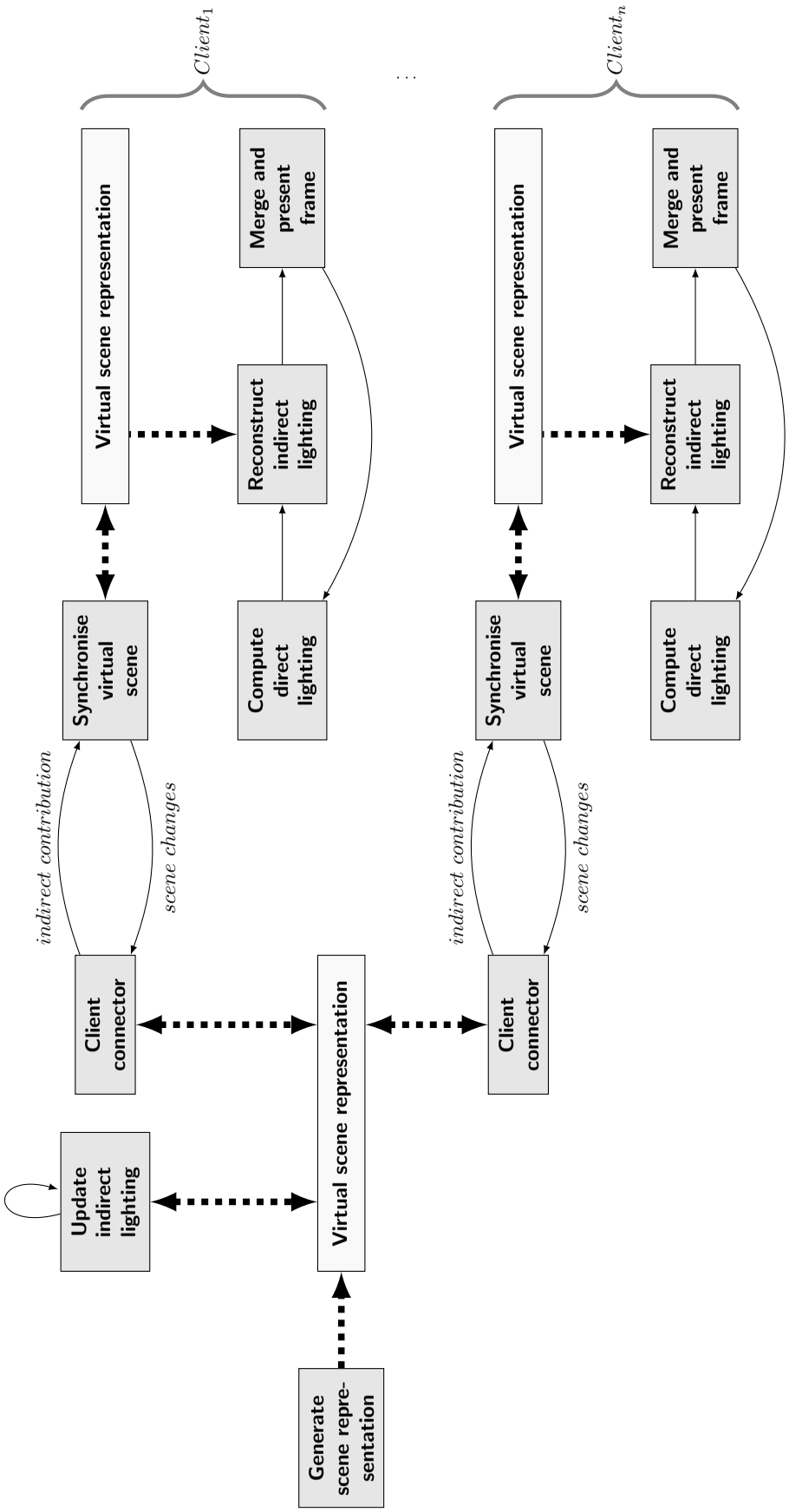


Figure 6.11: Rendering using remote asynchronous computation, high-level architecture

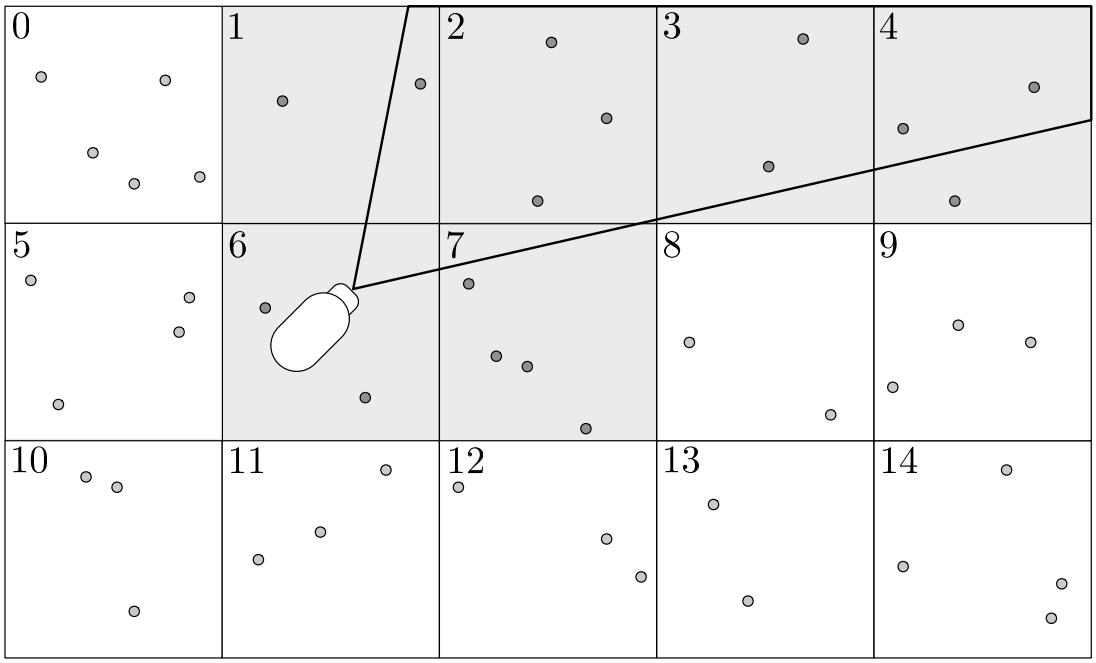


Figure 6.12: Irradiance samples clipped by view frustum. Any grid cell that intersects a client's view frustum is fully included in the server's response message to an indirect lighting update request. Including the entire cell eliminates the need to test individual samples against the frustum and index each sample using a key-value pair, since the position in the list implicitly serves as a key.

transfers between the server and a client. Messages are packed using an LZF compressor (Lehmann, 2014), for a reduction in size of up to 40%.

Although the indirect lighting synchronisation process is asynchronous, it runs at a lower frequency than the client rendering. Moreover, in some scenes it can be very difficult to generate paths between light sources and the camera (Bashford-Rogers *et al.*, 2013), requiring a larger number of VPLs to be traced for a more accurate estimation of indirect lighting (Dammertz *et al.*, 2010). This situation is exacerbated when the light sources are moving and any accumulated contribution has to be constantly reset, leading to artefacts and flickering. To reduce flickering and the disparity between direct and indirect lighting update rates, a simple two-stage smoothing mechanism is used.

The first stage uses exponential smoothing to combine the current irradiance values in G_c with the newly received values from G_s such that for every sample, $E'_c = w_c E_c + (1 - w_c) E_s$, where E'_c is the updated irradiance value, E_c is the previous value held in the client cache G_c, and E_s is the updated value for the same sample. The weight w_c determines how quickly the local cache transitions

to the server version. Increasing w_c exploits temporal coherence and reduces artefacts introduced by sudden illumination changes, making for a smooth transition. However, as a side effect of the slower transition, indirect illumination may be perceived to be lagging behind the direct illumination. The weight w_c can be adjusted at runtime and is typically initialised to 0.5. An attempt has been made to base w_c on the change in orientation of the observer, with large changes resulting in a smaller weight and vice versa, and although preliminary results look promising this has not been formally evaluated.

The second smoothing stage is used to compensate for the difference in update frequencies between direct and indirect lighting. For an arbitrary sample, the irradiance value used in the previous frame to reconstruct indirect lighting (E_d) and the most recent update for that same sample (E_c) are linearly interpolated to give the impression that irradiance is updating at the same rate as direct lighting. Let the $E_r(t)$ be an interpolation function:

$$E_r(t) = \begin{cases} E_d & t \leq 0 \\ E_r(0) + \dfrac{t}{\Delta T}(E_r(\Delta T) - E_r(0)) & 0 < t < \Delta T \\ E_c & t \geq \Delta T \end{cases} \tag{6.9}$$

where $t \in [0, \Delta T)$ is the time elapsed since the last cache update ($G_c = G_s$), ΔT is the interval to the next cache update; the frequency of cache updates determine how quickly E_d approaches E_c. The interval length to the next cache update, ΔT, is not known a priori, therefore it is estimated using an exponential average function:

$$\Delta T_{n+1} = w_d \ \Delta T_n + (1 - w_d)\tau_n, \tag{6.10}$$

where τn is the recorded interval for the nth update, ΔT_n is the estimated interval for the nth update, and $w_d \in [0, 1]$ determines whether more weight is given to the previous estimate or the actual reading when computing the next estimate. For $w_c = 1$, the next estimate is based entirely on previous estimates, $\Delta T_{n+1} = \Delta T_n$; for $w_d = 0$, the next estimate becomes the length of the last recorded interval, $\Delta T_{n+1} = \tau_n$. ΔT_0 is set to 100 (ms). In an ideal scenario where no network and communication fluctuations are present, a small value of w_d will quickly converge to a constant update interval. Nevertheless, even on an ideal network, the messages exchanged by client and server are still expected to vary in size since the amount of data transferred is proportional to the number of irradiance

samples contained in the view frustum of the observer (see §6.3.1). Empirically, values of w_d in the interval $[0.55, 0.8]$ were found to give the best results and produce less variance in the estimations, in the general case (see §6.4.1).

6.3.2 Amortisation of Computation

The virtual scene representation used for rendering is available on both the server and the connected clients. The server needs this information to be able to compute indirect lighting, while the clients require the scene for local rendering, including the reconstruction of indirect lighting and possibly any additional application logic that manipulates the objects within. A client that changes the scene representation by moving objects or lights, for instance, is responsible for informing the server (see Figure 6.11) and initiating the synchronisation process to ensure the scene is consistent at both ends. The point set Q, which is the representation of diffuse indirect lighting in the virtual scene, is updated once for all connected clients that share the same multi-user environment. Notwithstanding any possible work replication due to scene synchronisation mentioned above, the centralised indirect lighting computation outweighs the penalties thereof; not only are these costs quickly amortised, but the more clients participating in the same virtual environment, the greater the benefits in terms of computation sharing, which one could argue, result in a form of speed-up.

6.4 Results

The following results address the scalability of the system, both at client and server-side, bandwidth requirements, latency, and the respective error incurred due to these networking constraints. The system has been tested on four scenes, Sponza Atrium (both the original version and Crytek's), Tony's Barbershop, a Half-Life 2 death match community map, and Conference Room (see Figure 6.13). The Barbershop data set was introduced because besides being a traditional interactive raster scene, it is densely populated with geometric detail from small objects. The generated point cloud representation sizes for the maps were 14.5K, 21.5K, 32K and 38.5K points respectively. The parameters for the production of these point sets were empirically determined to strike a balance between quality and performance, in terms of both computation and communication (see Table 6.1). The hardware platform employed as a server for these experiments

(a) Sponza Atrium (Crytek)

(b) Conference Room

(c) Tony's Barbershop

(d) Sponza Atrium (Dabrovic)

Figure 6.13: The scenes used for remote rendering using asynchronous computation.

is equipped with two Intel Xeon E5-2697 CPUs (12 cores per processor), 64GB of RAM and an NVIDIA GeForce GTX Titan. The client setups range from an ASUS Transformer T100 tablet, equipped with an Intel Atom Z-3740 processor (clocked at 1.33GHz) and 2GB of RAM, to an Intel Xeon E5-2643 (4 cores per processor), 16GB of RAM and an NVIDIA GeForce GTX 680 display adapter. The server-side implementation uses NVIDIA OptiX to accelerate VPL shooting and occlusion testing. The client-side implementation has been carried out in Unity3D; reconstruction shaders were written in Direct Compute and thus require Direct3D11-capable hardware.

Scene	**Points**	**Min %**	**Max %**	**AO %**	**Sources**	**Darts**
Barbershop	32.0K	0.25	10.00	10.00	1024	256
Conference	38.5K	0.25	10.00	10.00	1024	256
Sponza Crytek	21.5K	1.25	25.0	25.0	1024	256
Sponza	14.5K	1.25	20.0	20.00	1024	256

Table 6.1: Parameters used in the generation of point sets. **Min** and **Max** determine the size of the minimum point distance and largest area of influence respectively, and are defined as % of scene size. **AO %** is the radius of the hemisphere of occlusion, **Sources** is the number of initial dart sources traced, and **Darts** is the number of darts thrown by each source.

6.4.1 Preliminaries

The values for two-stage smoothing (see §6.3.1) were set to $w_c = 0.5$ for the first and $w_d = 0.65$ for the second stage respectively. The value for the second stage was determined experimentally; a sequence of one hundred cache update intervals $\tau_i \dots \tau_{i+99}$ was sampled and the estimation error $(\Delta T_n - \tau_n)$ was recorded for $w_d \in \{0, 0.05, 0.1, \dots, 0.95, 1\}$. To simulate fluctuations both in the network and communication, the recorded intervals were modulated by a sinusoidal function:

$$\tau'_n = \tau_n \left(\sin \frac{2n\pi}{s} X_n + 1 \right), \tag{6.11}$$

where $s \in \{16, 32, 64\}$ determines the period of the sinusoid, X is a uniformly distributed random sequence with elements in the range $[0, 1)$, and τ_n is the n^{th} recorded interval. Figures 6.14, 6.15, 6.16 and 6.17 show the interval estimates plotted against the actual intervals, and the respective error for each value of

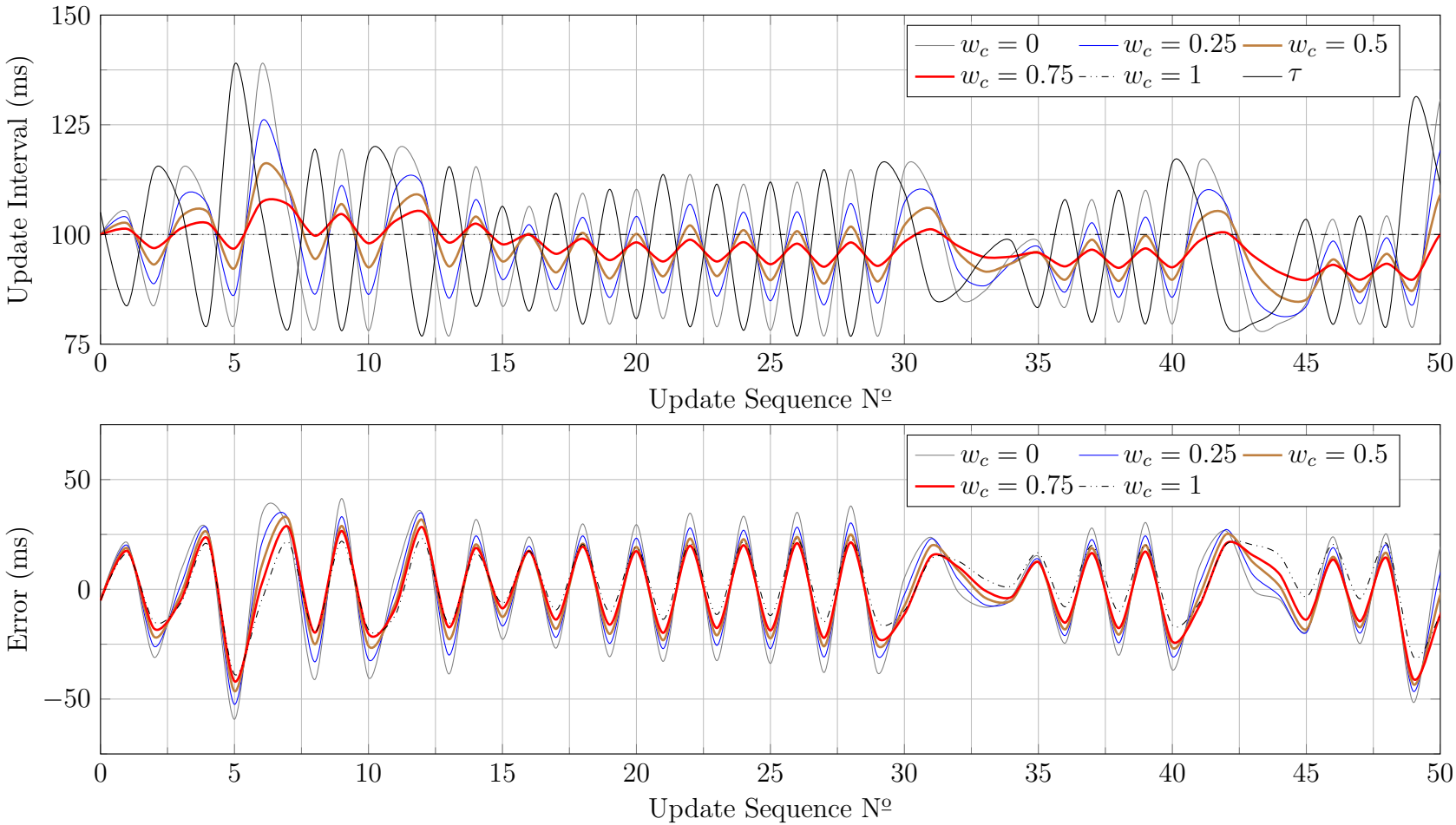


Figure 6.14: Recorded cache update intervals versus predicted intervals. No sinusoidal modulation.

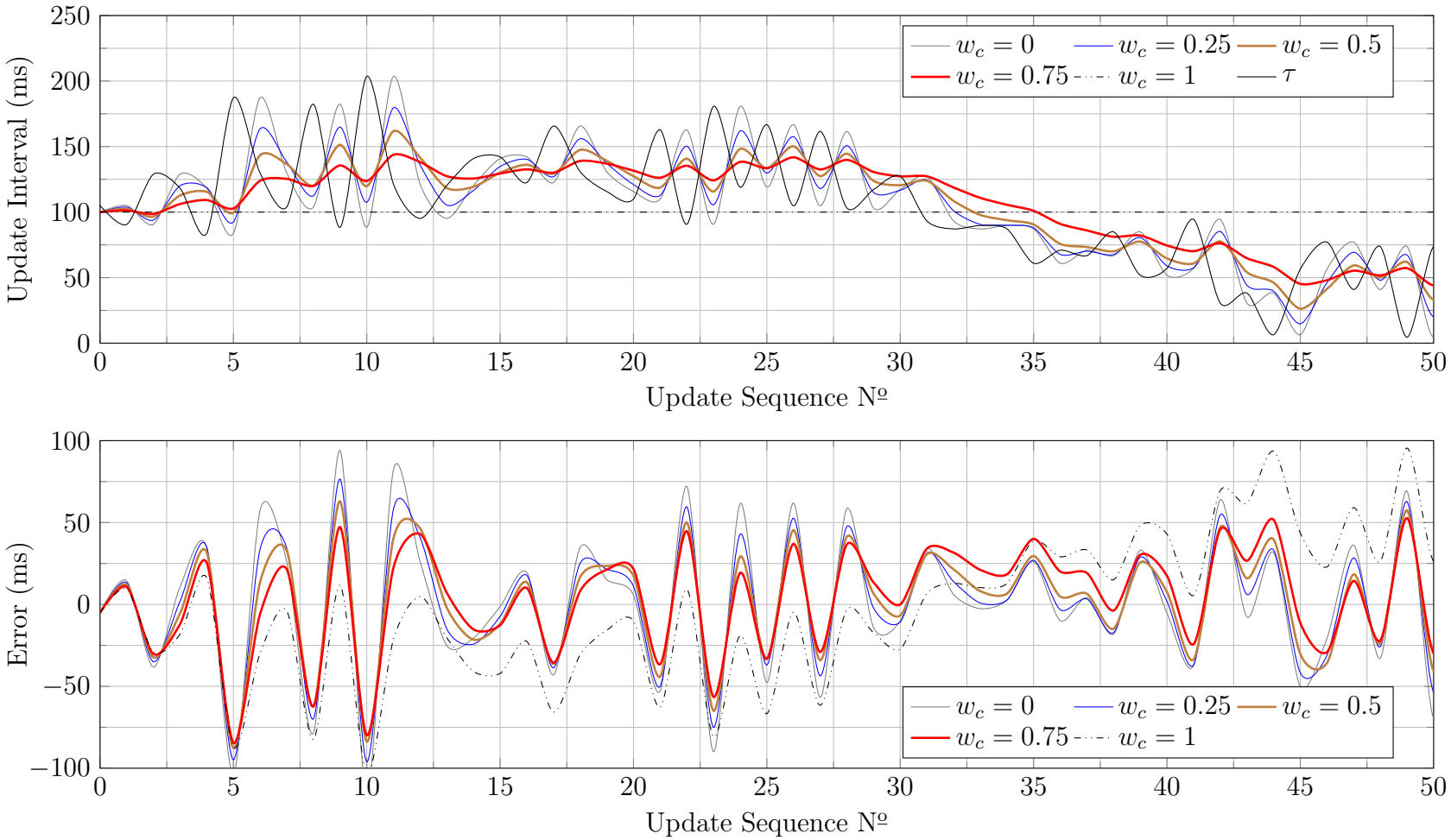


Figure 6.15: Recorded cache update intervals versus predicted intervals. Recorded intervals are sinusoidally modulated, with $s = 64$.

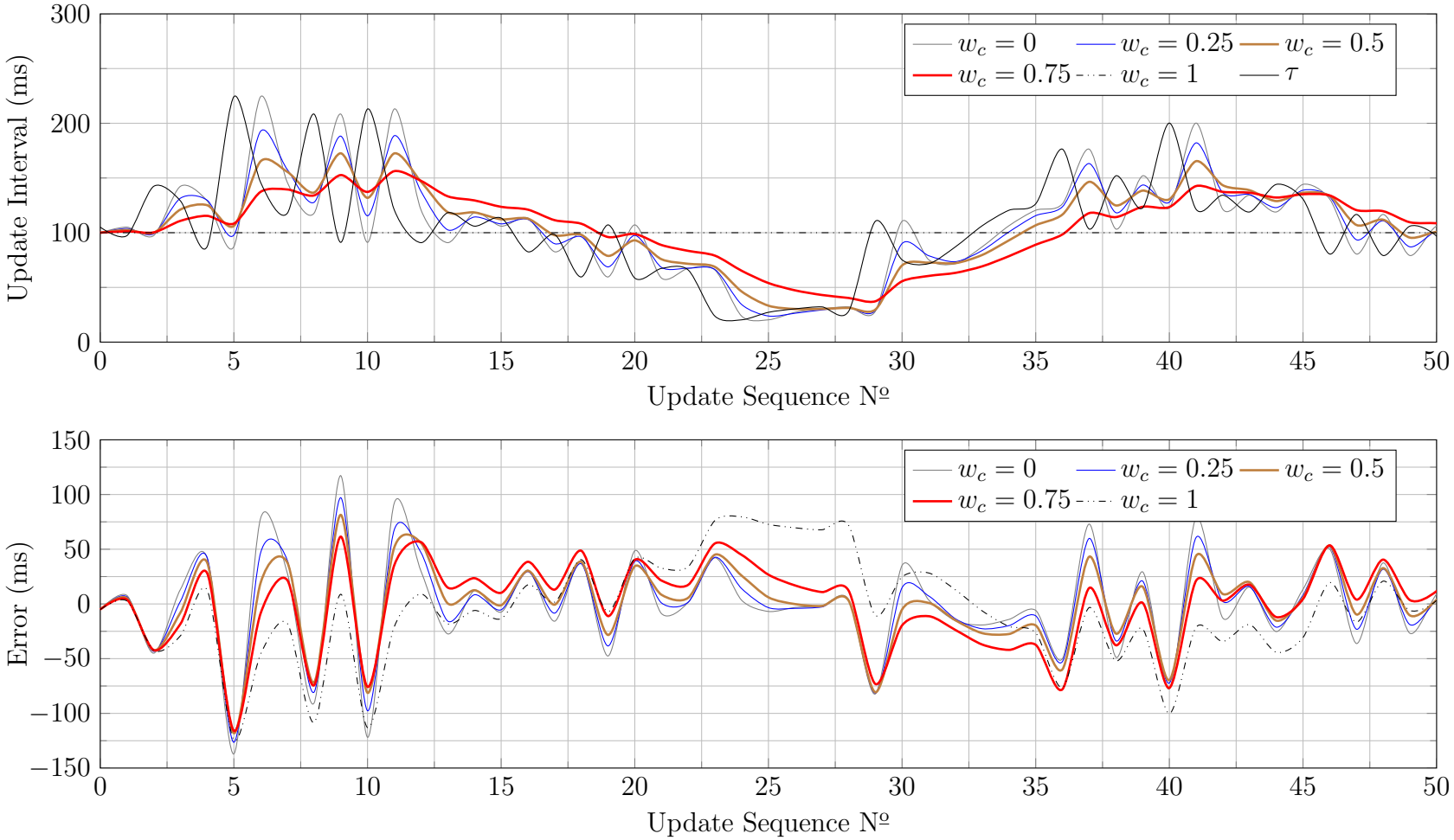


Figure 6.16: Recorded cache update intervals versus predicted intervals. Recorded intervals are sinusoidally modulated, with $s = 32$.

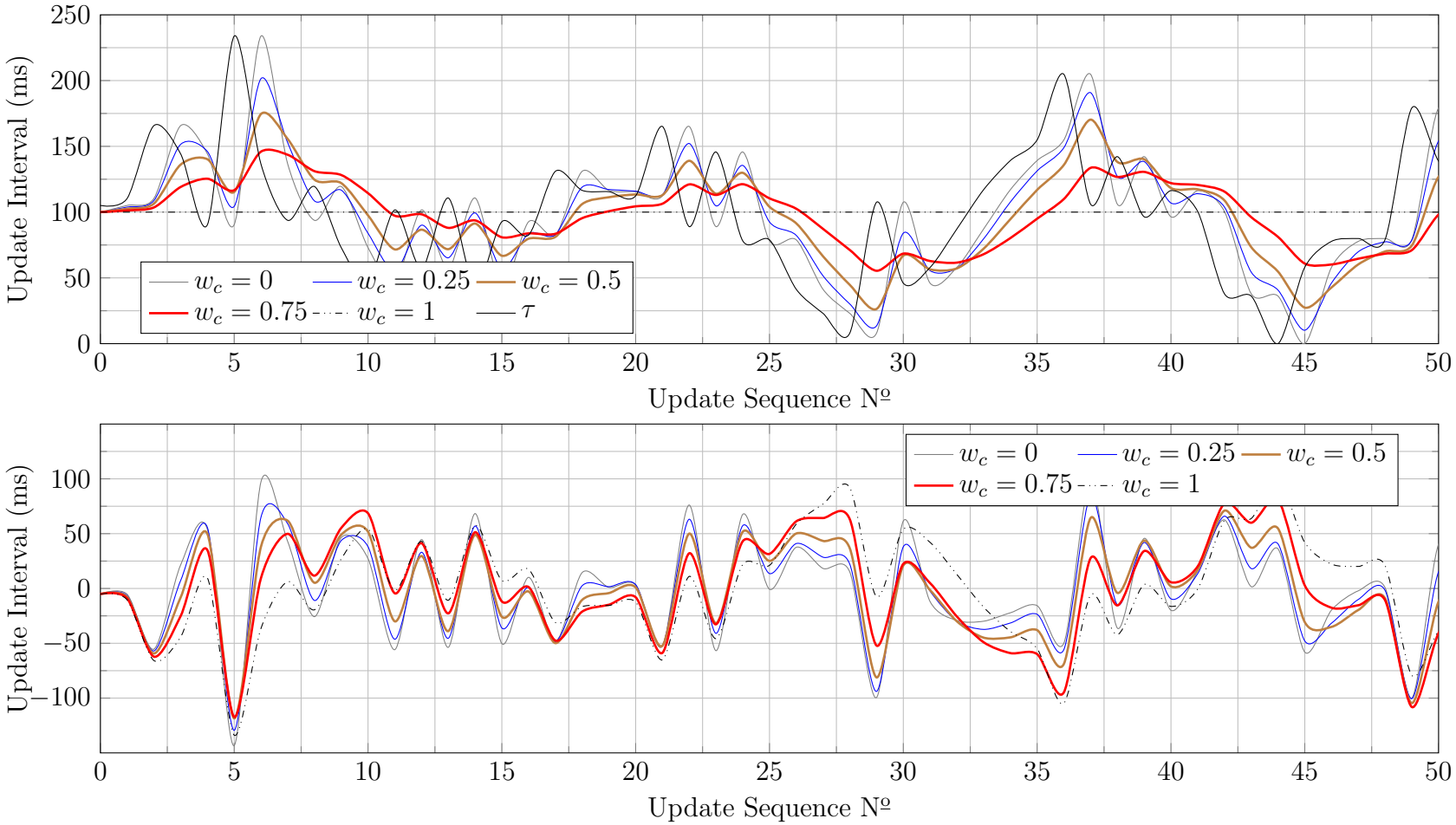


Figure 6.17: Recorded cache update intervals versus predicted intervals. Recorded intervals are sinusoidally modulated, with $s = 16$.

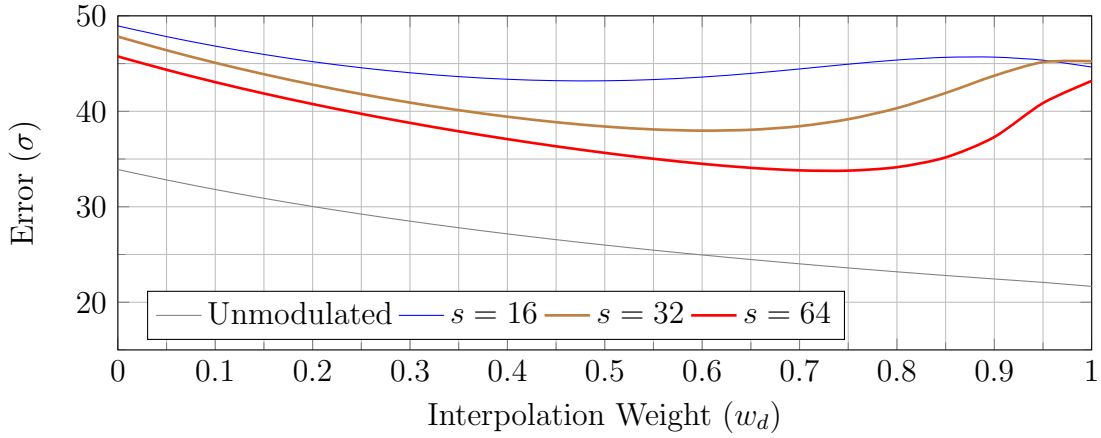


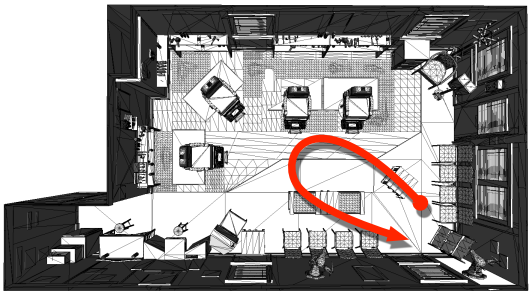
Figure 6.18: Standard deviation for prediction error.

w_d. In particular, 6.14 shows the predicted values over the sequence as originally sampled, while 6.15, 6.16 and 6.17 show the modulated sequence using values of 64, 32 and 16 for s, respectively. Figure 6.18 shows the variance in the error for the interval sequences. From the graph, it can be seen that the interpolation weights yielding less variance on average lie in the range $[0.55, 0.8]$. Taking the mean of the error values for each individual weight in the range returns the lowest value at $w_d = 0.65$; this value is used in the following results.

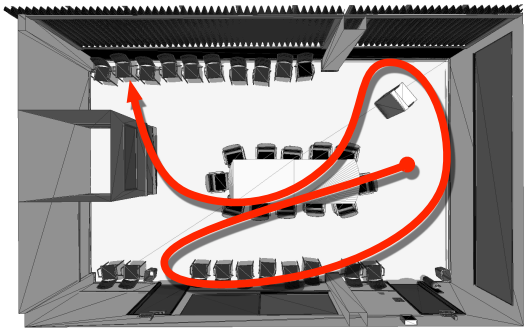
The bandwidth (§6.4.2) and image-fidelity (§6.4.4) results have been recorded over a scripted set of paths, one for each of the four scenes, since it was necessary to replicate the camera movement over multiple runs of the experiment. The walkthrough paths are shown in Figure 6.19.

6.4.2 Bandwidth

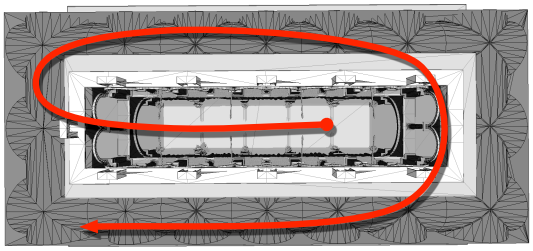
The bandwidth requirements of the system have been recorded for all test scenes (see Figure 6.20). Both the camera view and the light sources followed a scripted path wherein they were constantly changing. Thus, the results of this test do not take into consideration the quiescence that may be achieved by scenes which change very infrequently or in bursts. The client and server complete a roundtrip exchange of indirect lighting at an average frequency of 6 Hz. The total bandwidth requirements, shown in the last column of Table 6.2, do not exceed 3 Mbps, which is on the same level as the minimum recommendations for game streaming services, but for the fact that our system is resolution invariant, and reconstructing UHD quality images requires no additional bandwidth. It is worth



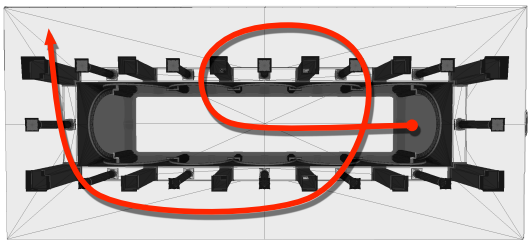
(a) Tony's Barbershop: the camera direction is fixed towards the north-west corner of the room.



(b) Conference Room: the direction of the camera follows the highlighted path.



(c) Sponza (Crytek): the camera is fixed vertically at half the height of the bottom set of columns. Camera view follows the highlighted trajectory.



(d) Sponza (Dabrovic): the camera is vertically aligned with the first floor and follows the highlighted path. The view follows the traced path.

Figure 6.19: Scripted walkthrough trajectories for tested scenes.

pointing out that although the Barbershop scene has a smaller point cloud than the Conference scene, it is nonetheless more densely populated. This results in a larger number of points captured by frustum culling when compared to the other scenes, which also reflects in the higher bandwidth requirements. Notwithstanding, these requirements are lower than most game streaming requirements for even 720p resolutions.

Scene	**Points**	**Mbps**
Barbershop	32.0K	2.826
Conference	38.5K	1.712
Sponza Crytek	21.5K	1.540
Sponza	14.5K	1.159

Table 6.2: Bandwidth requirements for the tested scenes.

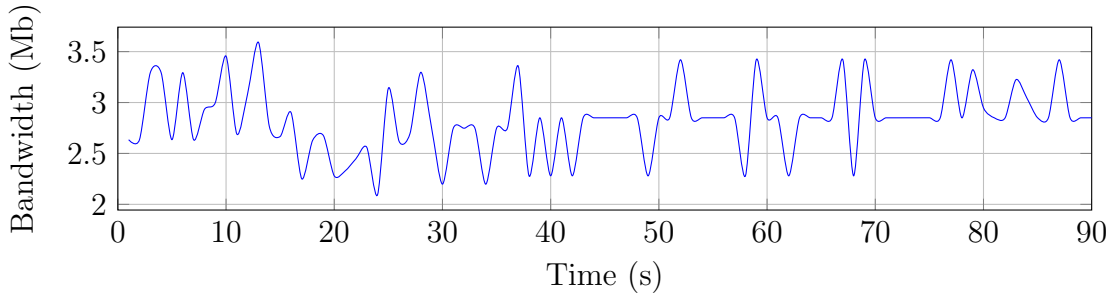
The results highlight the potential of achieving high-fidelity graphics on resources of varying computational power without compromising interactivity re-

sponse times due to network fluctuations and bandwidth constraints. The method scales adequately at both ends of the hardware spectrum, on average achieving frame rates of approximately 25 Hz at a resolution of 1024×768 on the Intel Atom tablet, and over 60 Hz at UHD resolutions on a desktop PC equipped with a GeForce GTX680.

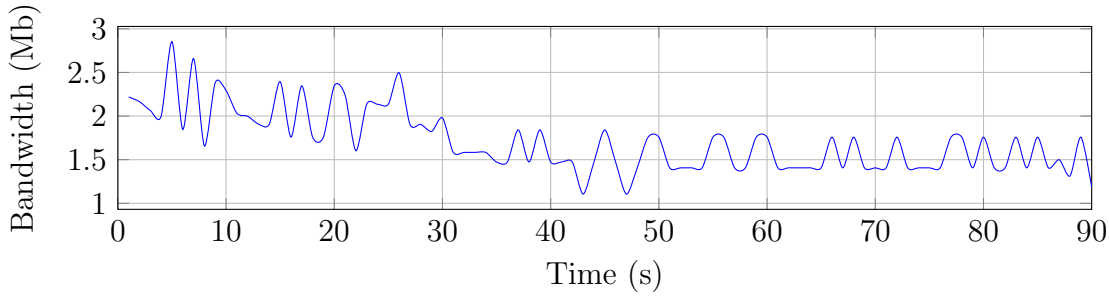
6.4.3 Client Scalability (Remote System Overhead)

System scalability is measured in the ability of the system to support multiple connected clients without service degradation. In particular, degradation manifests in an increased communication latency between clients and the server-end, when indirect lighting is being streamed to the former. Updates to indirect lighting follow a request-response model, where the client initiates each update itself; this request-response cycle has been measured for all test scenes with a varying number of clients and it was found that the increase in latency for up to 24 clients is almost negligible, suggesting that the system is capable of scaling well beyond this number (Figure 6.21).

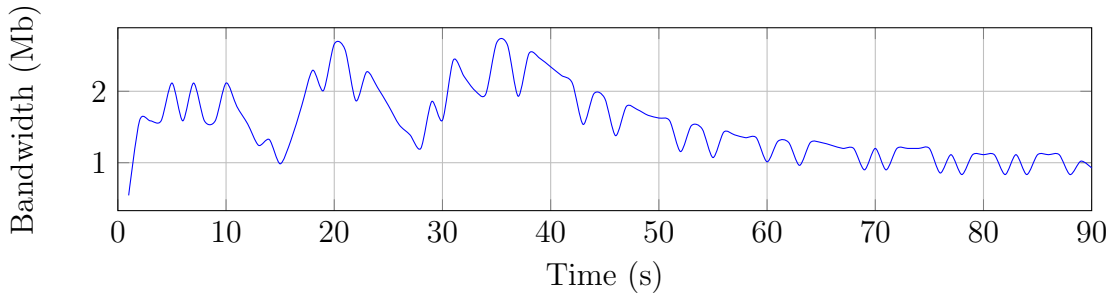
Figure 6.22 shows the amortisation of computation for the indirect diffuse component, in each of the four scenes. If the computation were to be decentralised and moved back to each individual client, for a homogeneous group of clients c where each member is working individually, the amount of work done would increase by a factor of $c - 1$. For t ms of original computation time on the server, the total work for c independent clients would increase to ct ms. In RAIL, indirect lighting computation is valid for all connected clients, and thus, the ideal computation time would be t as opposed to ct. However, Figure 6.21 highlights the fact that realisation penalties exist that are introduced due to communication and synchronisation, and increase as more clients are added. To account for them, the actual computation time for c clients becomes $r_p(c) + t$, where $r_p(c)$ is the realisation penalty for c clients. Figure 6.22a, 6.22b, 6.22c and 6.22d show plots of t (Ideal), $r_p(c) + t$ (Par.) and ct (Seq.) for all four scenes. Figure 6.22e expresses this gain in terms of computation speed-up; it must be stressed that the gain comes at no cost since the clients would otherwise still have to perform the computation of indirect lighting themselves.



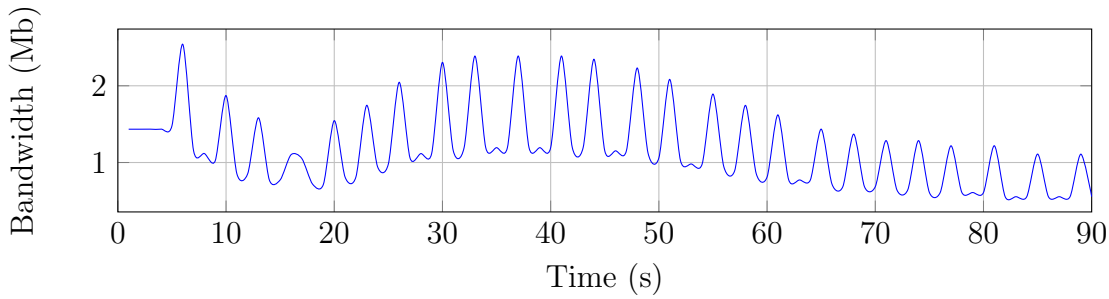
(a) Tony's Barbershop



(b) Conference Room



(c) Sponza (Crytek)



(d) Sponza (Dabrovic)

Figure 6.20: Bandwidth values for animation sequences over Sponza Crytek, Dabrovic, Barbershop, and Conference Room respectively.

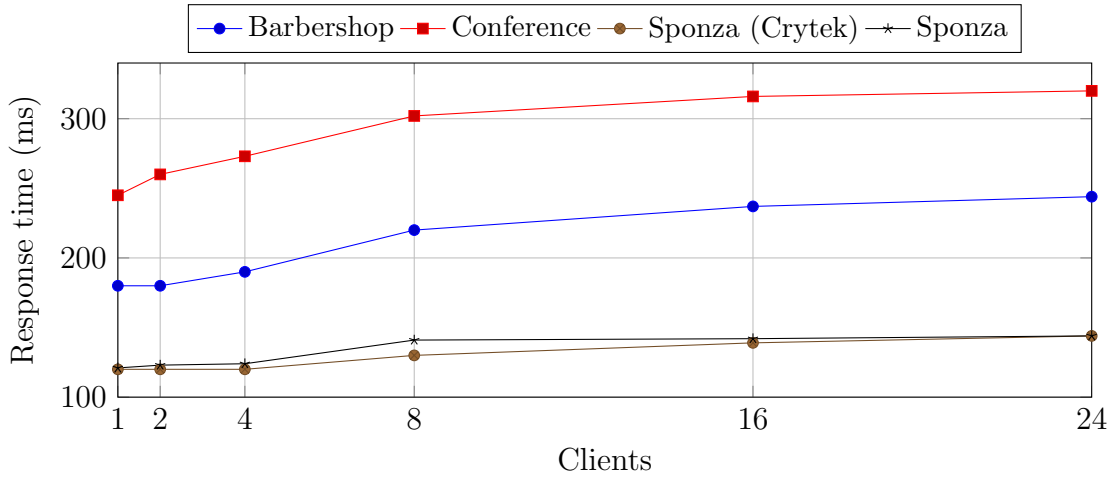


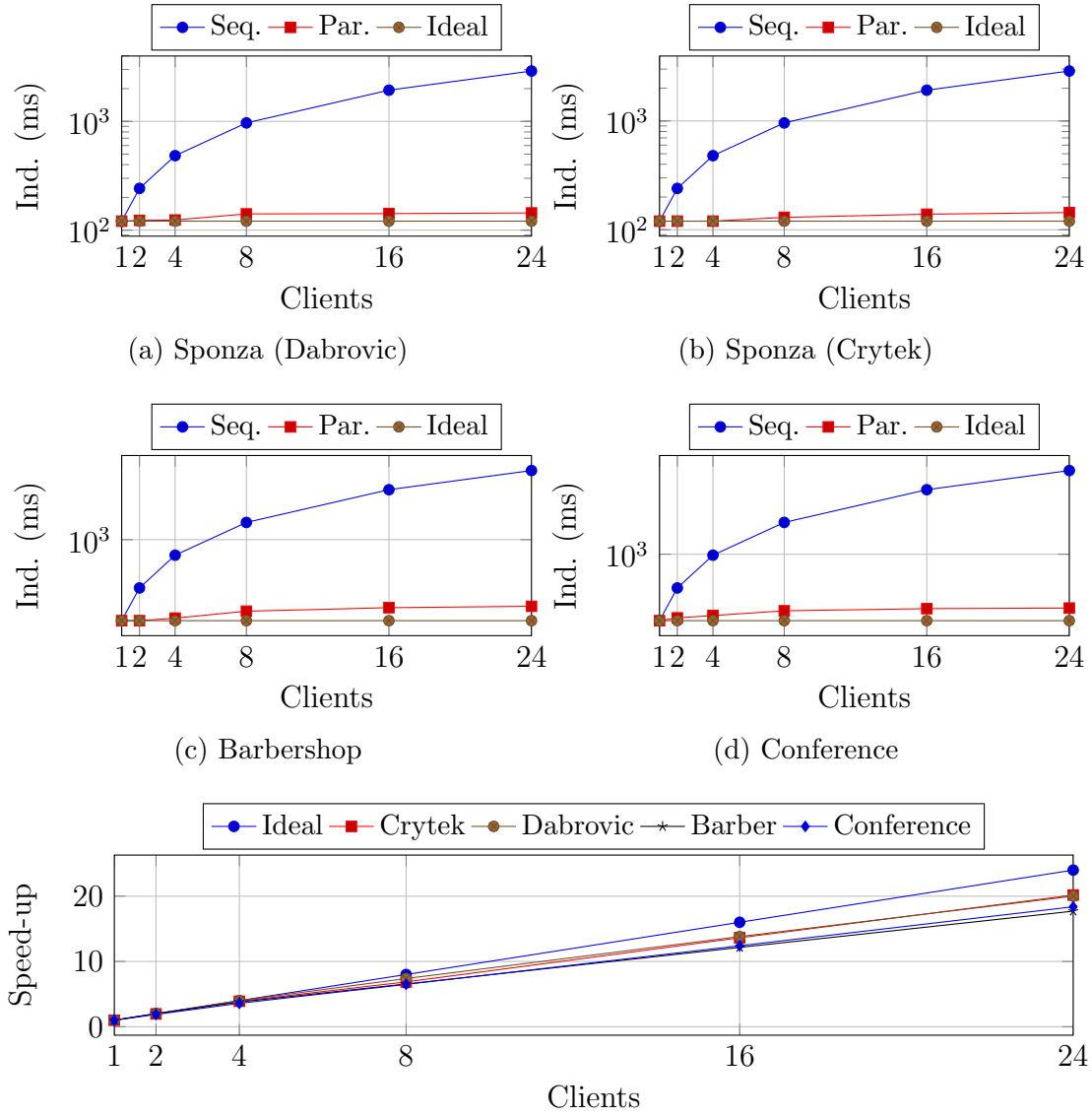
Figure 6.21: Changes in response times for tested scenes as the number of clients increases - a measure of scalability.

6.4.4 Image fidelity

The remote asynchronous nature of RAIL introduces temporal discrepancies between the direct and indirect lighting components. In this test we measure image fidelity as a function of these discrepancies; particularly, we measure the difference between a typical and a zero-latency execution of the system, the latter generated entirely and synchronously on the server using the rendering method described in §6.2, without any time constraints. For both methods, scripted walkthroughs over three of the test scenes were rendered and compared (see Figure 6.19). The PSNR was computed for each pair of frames in the resulting animations and is shown in Figure 6.23. The light sources and camera view change throughout all but the end of the animation, where the image was allowed to converge over a number of frames. The effect of this convergence is evident in Figure 6.23, as the PSNR increases towards the end of the sequence. In order to put the PSNR values in context, multiple zero-latency runs of the same walkthrough were rendered and compared against each other by computing the PSNR of the resulting frames. The average PSNR for these sequences was 36.

6.5 Discussion

The results highlight the potential of achieving high-fidelity graphics on resources of varying computational power without compromising interactivity response times due to network fluctuations and bandwidth constraints. Server-side scal-



(a) Sponza (Dabrovic)

(b) Sponza (Crytek)

(c) Barbershop

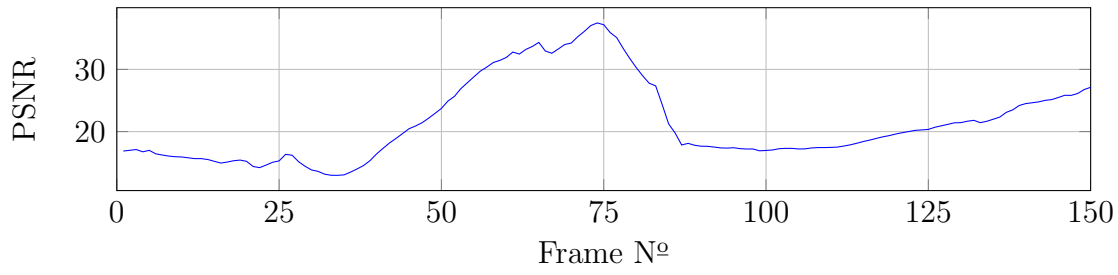
(d) Conference

(e) Speed-up due to amortisation of indirect lighting.

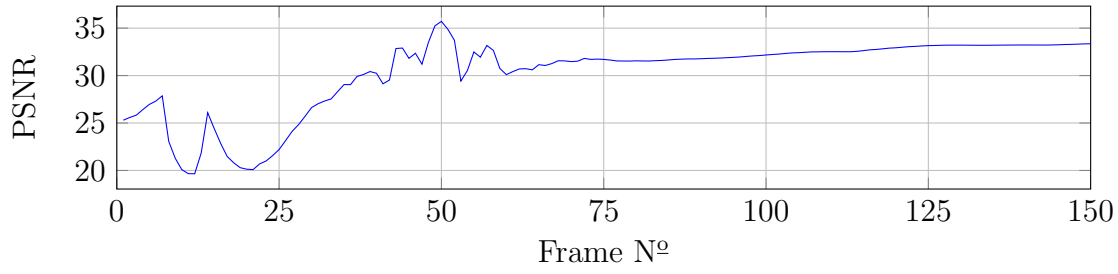
Figure 6.22: Computation gains due to amortisation of diffuse indirect lighting.

ability is very promising, with minimal overhead incurred when increasing the number of clients. The scalability results in Figure 6.21 show that additional clients in multi-user environments can be added at very little cost, since indirect lighting is amortised over them. This carries a significant advantage over streaming solutions which provide each of the clients with rendering sandboxes that do not interact and thus, share no computation load.

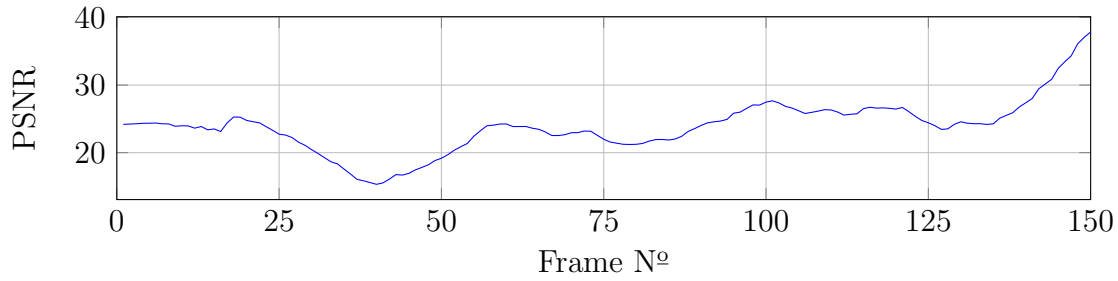
In CloudLight, Crassin *et al.* (2013) propose three lighting algorithms for different bandwidth and client configurations. Two of these algorithms, the



(a) Tony's Barbershop



(b) Conference Room



(c) Sponza (Crytek)

Figure 6.23: PSNR values characterising latency of indirect lighting over animation sequences for tested scenes.

path-traced irradiance maps and real-time photon mapping, parallel the approach used in RAIL by decoupling client updates from the cloud computation and the network performance. Irradiance maps yield low bandwidth requirements, and reconstruction costs are also cheap, but the difficulty in acquiring UV-parameterisation for moderately complex scenes doesn't always make them a viable option due to the laborious nature of the parameterisation (Crassin *et al.*, 2013). In contrast, the precomputation step in RAIL can be fully automated. Photon tracing doesn't require any parameterisations but has substantially larger bandwidth requirements, close to an order of magnitude more than the requirements of RAIL. Moreover, the indirect lighting reconstruction at the client poses prohibitive computational costs for some low to mid-range devices. The third

algorithm, which adopts a synchronous approach, uses cone-traced sparse voxel global illumination, and although client updates at 30 Hz can be sustained for 5 clients, this soon drops to 12 Hz as soon as the number of clients is increased to 24.

For the generation of the shading point clouds, the algorithm based on Bikker & Reijerse (2009) was adequate to demonstrate the concept of remote asynchronous rendering. Nevertheless, RAIL is not tied to a specific generation algorithm and more efficient ones may be used if so desired. As a note on the generated sizes of point sets, an overly dense data set may result in higher bandwidth usage as well as longer computation times at the server end, for the estimation of indirect lighting. This is suggested by the bandwidth usage (Table 6.2) and the request-response cycle times (Figure 6.21). Even though the Barbershop point set is smaller than that of the Conference room, it doesn't perform as well, due to its higher density.

While only support for multiple-bounce diffuse indirect lighting has been presented, other aspects of high-fidelity graphics such as glossy reflections, subsurface scattering and participating media could also be supported on the server via a similar method. Indirect lighting is characterised by shadows cast by secondary lights (or indirect shadows). In most popular GPU single-bounce algorithms, this phenomenon is mostly missing. RAIL simulates multiple bounces of indirect lighting and correctly captures the effect of indirect shadows. This is shown in Figure 6.24 where the light from the spotlight which bounces off the wall is blocked by the columns, as can be observed from their shadows on the floor.

6.6 Summary

In this chapter we have presented RAIL, an original method for efficient and scalable rendering by decoupling the expensive indirect lighting computation from the rest of the solution. The results demonstrate that it is possible to compute high-fidelity graphics on devices of lower computational power. While other cloud methods have been presented, both as fully streaming solutions and as distributed rendering pipelines, our solution requires lower bandwidth and is robust to latency. Furthermore, with respect to purely streaming solutions, our method can amortise the computation of indirect lighting in multi-user environments with minimal costs for each additional client (see Table 9.3). In contrast to

Figure 6.24: Columns cast indirect shadows from light bouncing off the wall.

RaaS, RAIL is hybrid approximative method, sacrificing quality and correctness for speed, even though it builds incrementally on the former. RAIL also underscores a very important point in the context of rendering as a service: the use of GPUs is indispensable in providing the acceleration required for ray tracing-based rendering methods, further validating the approach of RaaS.

CHAPTER 7

Precomputed Per-vertex Indirect Lighting (PPIL)

An accurate lighting model is one of the most important factors in conveying realism, and a desirable property of any photorealistic rendering system (Myszkowski *et al.*, 2001). In Chapter 6 it was shown that with the aid of a powerful back end, high-fidelity graphics can be achieved on low-end devices. In the absence of a powerful server, precomputation of lighting is a viable solution, and although some constraints are enforced on the nature of dynamic lighting, it can still help achieve otherwise unattainable realism on such devices. For example, complex illumination may be stored in special textures known as lightmaps (Blythe *et al.*, 1999), where a compatible lighting model is applied at a preprocessing stage and the results cached within. At runtime, lightmaps are then blended with detail textures to augment the rendering. Lightmaps minimally affect performance, but they cannot be used where geometry or environment layout is generated at runtime. In multi-user virtual environments, the limited device capabilities of single participants may not permit the generation of lightmaps or any other form of precomputed lighting at runtime. Yet, when the participants are aggregated, the required resources may become collectively available. This chapter presents a method for low-end devices that precomputes per-vertex indirect lighting (PPIL) for static multi-user environments that are generated online. The requirement for a powerful centralised machine is waived, as the lighting computations are borne by the participating client devices themselves. The multi-bounce indirect illumination information generated is then used for interactive real-time rendering, with the costs thereof being comparable to rendering the scenes using a constant ambient term to account for indirect lighting. The dynamically-generated envi-

ronments and the respective indirect lighting remain static throughout the rest of the simulation.

The chapter is structured as follows: §7.1 introduces the chapter and outlines the respective contributions, §7.2 gives overview of how PPIL adapts RAIL for low-end devices, §7.3 discusses distributing the precomputation of indirect lighting, §7.4 and §6.5 demonstrate results from PPIL followed by a discussion and §7.6 concludes the chapter.

7.1 Introduction

Real-time global illumination algorithms such as Dachsbacher & Stamminger (2005), Ritschel *et al.* (2008) and Kaplanyan & Dachsbacher (2010) may be too computationally expensive for low-end devices such as entry-level smartphones. As a result, indirect lighting information is usually precomputed offline and merged with direct lighting at runtime, for example, by storing it on the textures. Procedural content generation and real-time content adjustment place demands on the creation of virtual environments such as the ability to generate content at runtime (Yannakakis & Togelius, 2011; Nygren *et al.*, 2011), which is not compatible with offline lighting precomputation. Furthermore, since no assumption can be made about the client device performing the rendering, online precomputation might not be a viable solution for a single device. In a multi-user virtual environment, on the other hand, precomputation may be amortised among the participating users. The GI algorithm introduced in RAIL, discussed in Chapter 6, is scaled down and applied as a precomputation step to allow a network of devices, even those with the most basic rendering capabilities, to perform high-fidelity rendering. The contributions of this chapter are:

- a scaled version of the global illumination algorithm in RAIL, adapted for per-vertex indirect lighting, that is suitable for devices with basic hardware rendering capabilities
- a distributed algorithm for precomputing indirect lighting information that is suitable for dynamically-generated environments

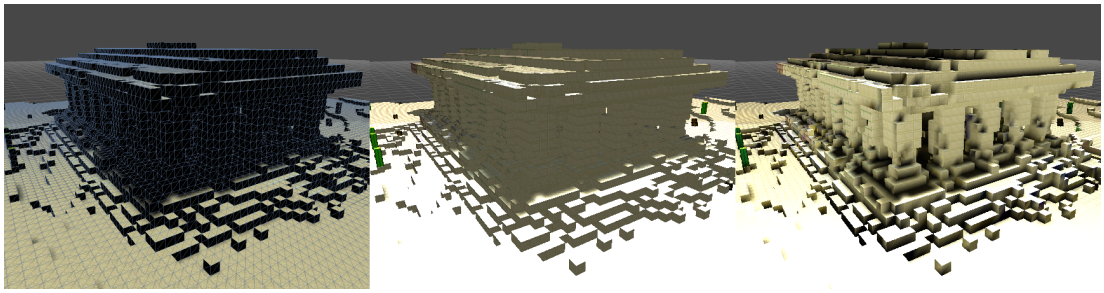


Figure 7.1: Mesh simulating a typical dynamically-generated world: wireframe mesh showing triangle tessellation (left), direct lighting with ambient contribution (middle) and our method simulating diffuse interreflections (right).

7.2 Method

PPIL is divided in two distinct phases, a precomputation phase, where the indirect lighting for the scene is evaluated, and the rendering phase, where the precomputed information is used to improve the quality of the interactive rendering. During the rendering phase, the dynamically-generated geometry and the respective precomputed lighting information do not change but remain static throughout.

7.2.1 Indirect Lighting Precomputation

Similarly to RAIL, the precomputation phase evaluates the indirect diffuse component for a point cloud Q that is representative of scene geometry, such that, for a sample point $\mathbf{q} \in Q$, the contribution of a set of VPLs with cardinality N is given by:

$$L_d(\mathbf{q}_p, \omega_o) = \sum_{k=1}^{N} f_r(\mathbf{q}_p, \omega_o, \omega_k)\ L_k\ V(\mathbf{q}_p, y_k)\ G'(\mathbf{q}_p, y_k), \tag{7.1}$$

where L_k is the emitted radiance of the k^{th} VPL, V is the visibility function between two points and G the bounded geometry term (Debattista *et al.*, 2009). Equation 7.1 is based on the area formulation of the rendering equation (see §2.5.2); for perfectly diffuse surfaces, where the outgoing radiance at $\mathbf{q}$ is uniform across the hemisphere Ω such that $f_r(\mathbf{q}_p, \omega_o, \omega_k) = \frac{\rho}{\pi}$, it simplifies to:

$$L_d(\mathbf{q}_p) = \frac{\rho}{\pi} \sum_{k=1}^{N} L_k\ V(\mathbf{q}_p, y_k)\ G'(\mathbf{q}_p, y_k). \tag{7.2}$$

7.2.2 Point Set Selection

A typical way of using Equation 7.2 in the context of rasterisation is that of evaluating the function for the required domain and storing the results in textures, in similar fashion to lightmaps. Particularly, $\mathbf{q}$ would map to a texel in a texture map that is in turn mapped to static geometry in the scene. The generation of these indirect lighting maps is not cheap, since Equation 7.2 must be evaluated for each texel, and is usually carried out offline. In general, this makes the technique unsuitable for providing indirect lighting to environments that are generated dynamically and cannot make use of fast and powerful machines to perform the said precomputation step. This is especially true in the case of weaker, less capable devices such as mobile phones and tablets.

The indirect lighting function L_d models a low-frequency function in space, as diffuse interreflections vary slowly on a surface. Thus, it can be sparsely sampled and interpolated with little loss in perceived quality (Ward *et al.*, 1988). In §6.2.2, a point representation Q of the scene had been generated in an offline process and the Shepard method used to reconstruct this indirect function for the whole scene (Shepard, 1968; Pál *et al.*, 2009). Although this precomputation step can easily be carried out at runtime, it is generally beyond the reach of low-end devices, which makes it unsuitable for dynamically-generated environments. Furthermore, even assuming that Q can be computed reasonably quickly, the runtime cost of executing a complex fragment shader that reconstructs indirect lighting from Q may not be viable. Nevertheless, let Q be the set of points at which L_d has been sampled; in order to extrapolate another sample point $\mathbf{x} \notin Q$ (in case $\mathbf{x} \in Q$, it follows that $\exists \mathbf{q} : \mathbf{q} \in Q \wedge \mathbf{q} = \mathbf{x}$, and thus $L'_d(\mathbf{x}_p) = L_d(\mathbf{q}_p)$), the following approximation is used:

$$L'_d(\mathbf{x}_p) = \sum_{\mathbf{q} \in Q} W(\mathbf{x}, \mathbf{q})\ L_d(\mathbf{q}_p), \tag{7.3}$$

where W is a weighting function that determines the contribution of each point $\mathbf{q} \in Q$, defined as:

$$W(\mathbf{x}, \mathbf{q}) = \frac{w(\mathbf{x}, \mathbf{q})}{\sum_{\mathbf{q} \in Q} w(\mathbf{x}, \mathbf{q})}. \tag{7.4}$$

The function w captures the geometric differences such as the angle between the surface normals and the distance between the points $\mathbf{x}$ and $\mathbf{q}$, similar to the irradiance interpolation function in the irradiance cache (Ward *et al.*, 1988). A

strategy for selecting Q in a deterministic way is that of using the vertex positions of triangles comprising scene geometry. The advantages of using such a sampling strategy for Q are, first and foremost, that the points are readily available and no additional data structures are required to store new information; secondly, storing the diffuse interreflection contribution L_d does not need existing rendering pipelines to be extended beyond the storage of an additional colour value per vertex. In contrast, RAIL uses a dart throwing technique to determine Q (see §6.2.1). The time complexity for sampling diffuse indirect lighting grows in the number of triangles, given Q maps to mesh vertices. More specifically, given t triangles comprising a scene, the worst case complexity is given by $\mathrm{O}\left(3t\sqrt[3]{t}\right)$, assuming triangles are stored in a tree-based acceleration structure (see 3.1). Figure 7.2 illustrates the precomputation stage where Q maps to mesh vertices.

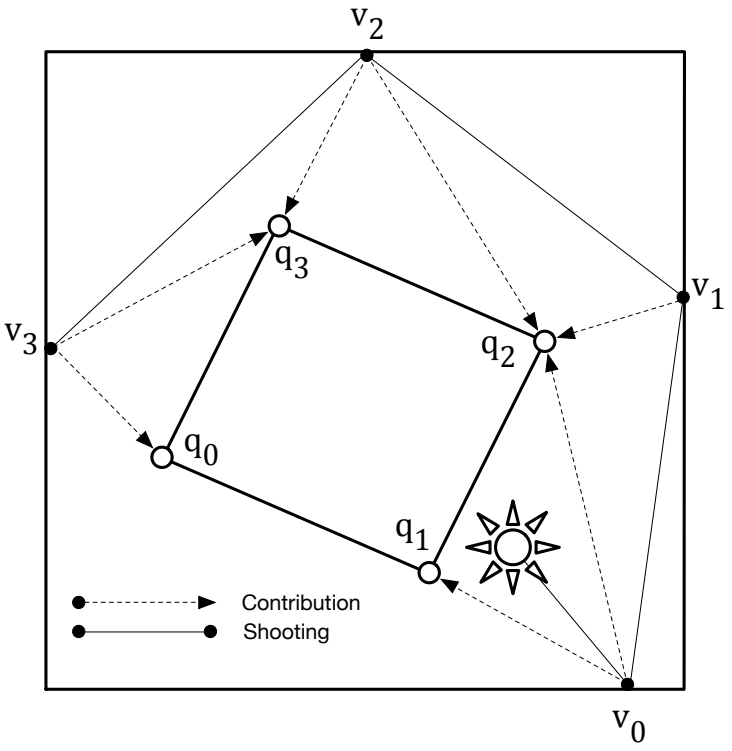


Figure 7.2: Shooting of point lights and computation of indirect lighting samples. In the first pass, VPLs v_0 through v_3 are created by tracing light in the environment. In the second pass, vertices $\mathbf{q}_0$ through $\mathbf{q}_3$ are tested against the VPLs and the respective irradiance computed.

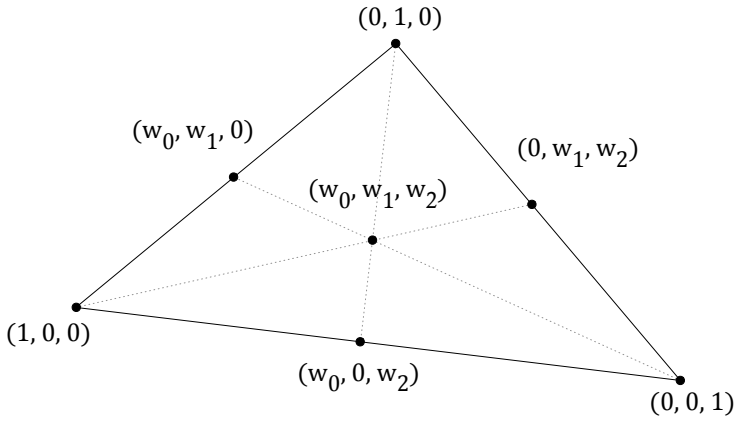


Figure 7.3: Barycentric coordinates to determine sample weight.

7.2.3 Rendering Phase

In §6.2.3, the reconstruction function employed the use of a regular grid to accelerate nearest neighbour searches of points in Q. Although this approach provides a reasonable speed-up over a brute force approach, especially when considering that most of the points in Q would contribute nothing to the final value, it may not be viable to implement on low-end devices. Thus, instead of considering a subset of Q whose element weights are greater than zero, the points comprising the triangle being currently rendered are considered. Equation 7.3 is rewritten in terms of a constant number of samples and a different weighting function; the weighting function is also simplified and bases the contribution of each of the three samples on the distance from the respective points in terms of the barycentric coordinates of the point being extrapolated:

$$L'_d(\mathbf{x}_p) = w_0 \; L_d(\mathbf{q}_0) + w_1 \; L_d(\mathbf{q}_1) + w_2 \; L_d(\mathbf{q}_2), \tag{7.5}$$

where $w_0 + w_1 + w_2 = 1$. This is illustrated in Figure 7.3, where a point on the face of a triangle is described in terms of its barycentric coordinates (w_0, w_1, w_2). In terms of implementation, each triangle vertex holds an additional colour value representing indirect diffuse lighting. When a triangle is rasterised, the three colour values at the vertices are interpolated and the result passed to the fragment shader, where it is added to the base lighting in similar fashion to a constant ambient term.

7.3 Distributing Computation

In Chapter 6, the shading of point cloud Q was computed in the cloud and the results shared with all the participating clients, amortising the computation over the connected users. In multi-user virtual environments where no such centralised computing power can be drawn from, the cost of shading Q can be amortised over the participating users by having each client compute part of the solution and sharing it with its peers. For an environment generation process that is entirely deterministic, the shading process can be carried out at each peer without the need to share large amounts of data. If the shading computation of each point $\mathbf{q} \in Q$ can also be guaranteed to be deterministic, it can then be partitioned and independently computed by peers in a distributed system. In particular, if the VPLs required for evaluating L_d differ on each machine, aggregating the results of distributed computations into a single data set could lead to discontinuities in the indirect contribution of adjacent surfaces, or in surfaces sharing contributions computed by different peers. To guarantee repeatability, shading computations, specifically the VPL shooting process, are seeded using quasi-random numbers. Low discrepancy sequences over the unit interval (Van der Corput, 1936) are employed, replacing pseudo-random numbers, to ensure that even though the process looks random, a good coverage of the sample space is still attained, while flickering and discontinuity effects that may arise due to peers using different sets of VPLs are eliminated.

The partitioning and distribution of computation follows a typical master-worker (bag of tasks) approach (see §4.2.2), whereby the elected master generates a list of independent tasks that may be computed in parallel, and makes them available to a set of workers, which consume them on a first-come first-served basis. In a client-server setting, the election of the master process is trivial since the server itself can act as a master. The strategy used throughout this study was that of electing the weaker machine in the group to act as master. The members of the group were known to all participants.

7.3.1 Master

The behaviour of the master process is shown in detail in Algorithm 9. Initially, the master partitions the unique geometry vertices in the scene into variable-sized subsets, to favour load balancing and ensure that work is more equally

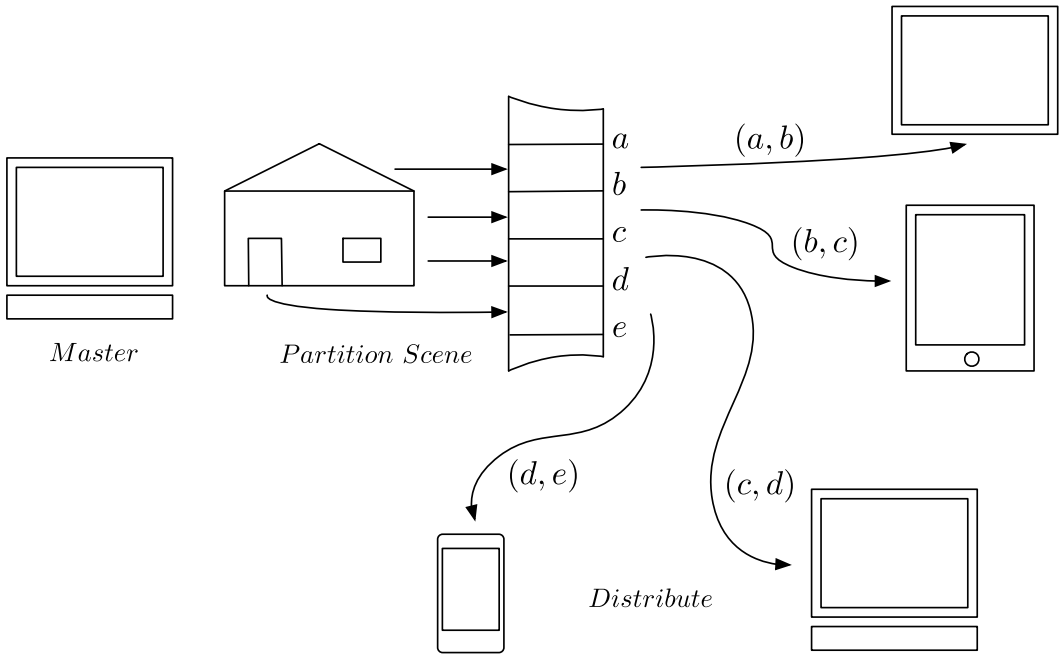


Figure 7.4: Partitioning and distribution of tasks. The elected master partitions the scene geometry

distributed among the workers in case of imbalances in job computation times of similarly sized jobs (Line 1). The master then proceeds to send tasks to available workers, marking the tasks as *taken* and the workers as *busy*, to avoid duplicate reassignments. For every task sent, a deadline is scheduled, in case the assigned worker fails to deliver results in time due to software or hardware failure (4-11). Any responses from busy workers are processed, aggregating the results with a central irradiance buffer and removing the completed task from the pending tasks list. The worker is marked as *idle*, so as to be included in the next round of task assignments (12-16). Tasks whose deadlines expire are unlocked and made eligible for reassignment to another worker. Also, workers that fail to keep with their deadlines are marked as *ignored*, and excluded from requesting further tasks (17-20). When all tasks have been completed and the results returned by the workers, the master sends a termination signal followed by the computed irradiance values to all workers (26-29). If all workers fail, the master aborts the computation; in this scenario, the application can resort to a fallback mechanism (22-24).

Algorithm 9 Master process algorithm for distributing shading tasks to workers

```
1: procedure MASTER(scene, workers)
2:     tasks ← subdivide(scene)
3:     while tasks.remaining ≠ 0 do
4:         for all {w : w ∈ workers ∧ idle(w)} do
5:             if tasks.pending then
6:                 t ← tasks.next                    ▷ lock and get next task
7:                 send(t, w)
8:                 tasks.lock(t)
9:                 workers.setBusy(w)
10:                deadlines.schedule(t)
11:            end if
12:        end for
13:        while receive(r) = true do
14:            workers.setIdle(r.sender)
15:            tasks.remove(r.task)
16:            scene.applyIrradiance(r.irradiance)
17:        end while
18:        for all {d : d ∈ deadlines ∧ expired(d)} do
19:            tasks.unlock(d.task)
20:            workers.ignore(d.worker)
21:        end for
22:        if workers.active = 0 then
23:            break
24:        end if
25:    end while
26:    for all {w : w ∈ workers} do
27:        send(terminate, w)
28:        send(scene.irradiance, w)
29:    end for
30: end procedure
```

7.3.2 Workers

Algorithm 10 illustrates the steps taken at the worker end to shade point cloud Q. As a first step, each worker traces the VPLs required for instant radiosity computations. At this stage, workers also initialise a timer to prevent getting stuck in the event of failures (Lines 2-3). Subsequently, each worker waits for a message from the master; the message could either be a task containing the points that require shading or a termination signal, which denotes the completion of the precomputation phase. Specifically, a received task contains two indices into the list of geometry vertices in the scene, which bounds the batch of points

Algorithm 10 Worker process algorithm for shading subsets of Q

```
 1: procedure WORKER(scene)
 2:     vpls ← tracePointLights(scene)
 3:     watchdog.start
 4:     while receive(r) ≠ terminate do
 5:         for r.start ≤ v ≤ r.end do
 6:             for all vpl ∈ vpls do
 7:                 E ← E + contribution(vpl)
 8:             end for
 9:             scene[v].irradiance = E/|vpls|
10:         end for
11:         result ← package(scene, r.start, r.end)
12:         send(result, master)
13:     end while
14:     receive(irradiance)
15:     scene.applyIrradiance(irradiance)
16: end procedure
```

the worker is expected to compute irradiance for. For each task, a worker iterates through the points in the batch and computes the contribution of each VPL (5-10). The results are packaged and sent back to the master process (11-12). When a termination signal is received by a worker, it is followed by the full set of irradiance values for all geometry vertices except those already in possession of the worker. When a worker fails a computation deadline, the master assumes none of the irradiance values of the respective task have been computed by the worker, notwithstanding. Having received the irradiance values, the worker updates the vertex structures (14-15). Although the tracing of VPLs is carried out independently at each worker, the repeatability of the process ensures that the generated point lights are consistent across all workers with any need for communication or synchronisation. Each path traced during the generation is bounded by a maximum number of bounces and may be terminated earlier via the use of Russian roulette ((Dutre *et al.*, 2003)). In the case of failure, a peer abandons the precomputation phase and moves on to the rendering phase using some fallback mechanism.

7.3.3 Communication

An irradiance sample is stored in 24-bit RGB format (8-bits per channel); messages are packed for transmission using an LZF compressor, for a reduction in

size of up to 40% (Lehmann, 2014). The transmission capacity required for an n-vertex scene with w participants is approximately $(1/c) \cdot 3n \cdot w$ bytes, where c is the data compression ratio. For example, in a scene of 300K vertices, a participant transfers on average 0.9 MB of uncompressed data (≈ 0.6 MB when compressed); for eight participants and a compression ratio of $c = 1.43$, the total data transfer in the system is approximately 5 MB.

7.4 Results

The scenes used to draw the following results have been chosen to be representative of indoor and outdoor scenes. The indoor scenes, Tony's Barbershop and Kitchen, are both Halflife community maps, while the outdoor scene, Temple, is a Minecraft community map consisting of a voxel landscape with a temple. The configuration of the scenes has been varied such that the general lighting conditions of the environment were affected; for instance, windows were opened or closed, light sources placed inside and outside the room, and furniture such as chairs moved around. These scenes have been chosen to simulate what happens in various procedural environments.

The results show that PPIL, besides being suitable for low-end devices, can capture the effect of diffuse interreflections, possibly providing a more immersive experience. Comparisons were carried out against a model that uses a constant ambient term to account for light scattering about the environment. SSAO was used to attenuate ambient lighting depending on how exposed each point is to it. Shadow mapping was used to simulate shadows from direct lighting in both models. The quality of indirect lighting and shadows, and rendering performance between these two methods are compared and contrasted. For the sake of brevity, the model against which comparisons are carried out is referred to as the *straightforward approach*. The test setup used ranges from Intel i5-based PCs to tablet devices like the Nexus 7 and 3^{rd} generation iPad; the precomputation engine is distinct from the real-time visualisation, the latter having been created in Unity3D. For the precomputation stage, the shading of the point set Q was carried out using 512 VPLs. The scene sizes range from 180K to 300K vertices.

Device	Scene	Straightforward	PPIL
Nexus 7 (1280 × 800)	Kitchen	18.0	17.2
	Barber	18.9	17.6
	Temple	32.3	31.4
iPad3 (2048 × 1536)	Kitchen	12.0	11.2
	Barber	13.5	12.8
	Temple	20.4	19.8

Table 7.1: Rendering speeds for the tested scenes in frames per second (Hz).

7.4.1 Timings

The rendering performance of PPIL was evaluated on two tablet devices, Android-based Nexus 7 and a 3^{rd} generation iPad, running on the iOS operating system. Table 7.1 shows the recorded runtime performance for both methods across the three scenes. The indirect lighting precomputation performance largely depends on the scene complexity, machine and network configuration. A single-threaded process running on an Intel Core i5 machine precomputes the indirect illumination for the Kitchen scene (270K) using 512 VPLs in approximately 70s. Unlike RAIL, the process does not make use of GPU computing to compute indirect contribution.

Running the precomputation phase in a distributed setting does not see linear speed-up because the machine elected to be the master does not do any computation. A network of three machines with similar specifications as above and multi-threading enabled can carry out the same computation in 10 to 12 seconds on average. Computation time under ideal speed-up would be $70/(3 * 4) \approx 6$ seconds, given that each machine has four processor cores. Factoring out the master, the computation time on two machines is $70/(2 * 4) \approx 9$ seconds, under ideal speed-up conditions.

7.4.2 Qualitative Comparison

In this section, a qualitative comparison of PPIL and the straightforward approach is provided. For each pair of images that are compared, a perceptually weighted visual difference of the images is provided (Mantiuk *et al.*, 2005), to highlighting the differences between the two methods. Difference images are colour coded using the scheme shown in Figure 7.5. Grey denotes no change.

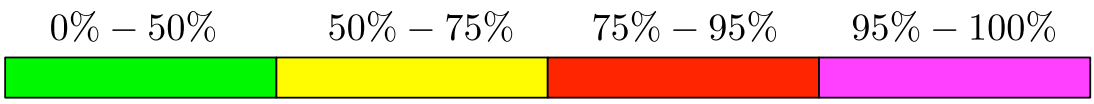


Figure 7.5: VDP difference scale

For some of the results, the change in contrast is also given, using the metric from Aydin *et al.* (2008). Here, the colour coded images denote a loss in contrast using green, an amplification using blue, and a reversal of polarity in red. Grey denotes no change.

Tony's Barbershop

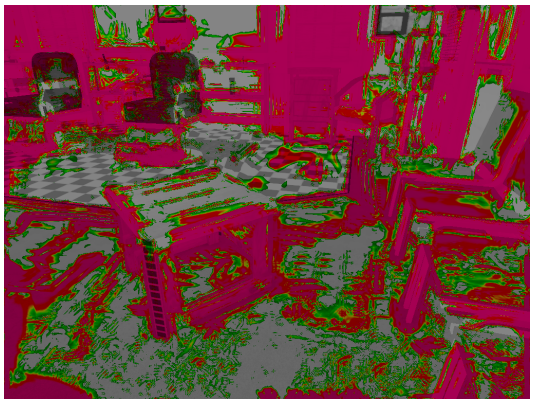
The first scene to be evaluated is Tony's Barbershop. Two configurations of the scene are presented. In the first, the scene is illuminated from light coming through an open window, on the right hand side of the scene. There are no light sources in the room itself, so the scene is mostly illuminated by indirect lighting. This is shown in Figure 7.6. In the straightforward approach, the constant ambient function is the dominant component illuminating the scene (Figure 7.6a), leading to a loss in contrast that would result from occlusion to indirect lighting. Figure 7.6b shows the view rendered using PPIL. Here, the effect of secondary bounces of light is evident in the localised colour bleeding of furniture on the grey walls, resulting in a reddish tint. Figure 7.6d highlights the overall change in contrast between the straightforward approach and PPIL, which appears to make objects more distinguishable. Figure 7.7 highlights the effects of diffuse interreflections between the furniture and the walls, resulting in colour bleeding that is missing in the straightforward approach.

The second configuration of the Barbershop uses closed windows and a point light source placed near the centre of the room. The point light simulates a tungsten bulb, and the illumination mainly consists of direct lighting, with very few occluded areas. The results from the second configuration are shown in Figure 7.8. In this figure, it can be observed that the differences between PPIL and the straightforward approach are not very marked and only present in regions that receive no direct light. This is to be expected since the direct lighting contribution is computed the same way in both methods. The difference images (Figure 7.8e and 7.8f) corroborate this observation and point out that the images diverge most in light-occluded and shadowed areas of the image.

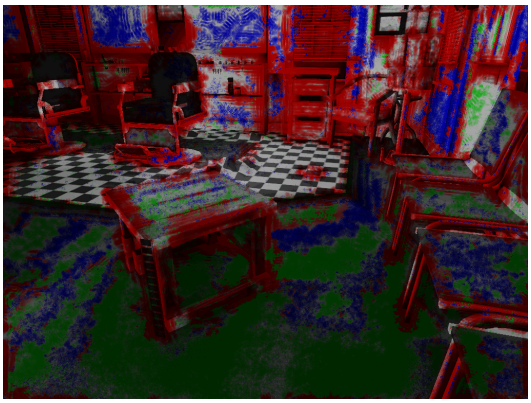
Indirect lighting is most noticeable in areas that are not directly lit. Figure 7.9

(a) Straightforward

(b) PPIL



(c) VDP



(d) DRIM

Figure 7.6: Tony's Barbershop scene, illuminated by light coming through window on the right hand side.

focuses on two regions of the Barbershop that are in shadow. Figure 7.9b shows yet another instance of colour bleeding, which is more apparent when compared to the straightforward approach (Figure 7.9a). Diffuse interreflections not only provide a natural colour bleeding effect, but also make shadowed areas appear less flat. Since the ambient function is constant across the whole scene, the regions in Figure 7.9c that are in shadow receive equal amounts of ambient lighting, even though some are more occluded than others. SSAO mitigates this problem but is generally limited to corners and edges and cannot detect more global features due to its screen-space nature. In Figure 7.9d, surfaces manifest various degrees of shadow tones, depending on their overall occlusion. This contributes to an overall increased perception of object shape and depth.

(a) Straightforward (b) PPIL

Figure 7.7: Colour bleeding due to diffuse interreflections in Tony's Barbershop scene.

Kitchen

The kitchen scene is also evaluated on two distinct configurations. The configurations follow the same pattern used in the Barbershop scene. In the first configuration, the light source is placed outside the room and a great part of the surfaces in the environment are lit from secondary bounces of light coming through the open window. The straightforward approach, shown in Figure 7.10a renders flat and dull looking surfaces. In the same view rendered using PPIL (Figure 7.10b), cues such as diffuse interreflections and soft shadows make the image look more plausible and realistic. Colour bleeding can be observed in the interior of the cupboard, the appliances and the floor, which reflects a hue of the furniture's material colour. The perceptual differences between the two methods (Figure 7.10c) is weakest on un-occluded flat surfaces. Figure 7.10d shows the difference in contrast between the two images, which mostly increases going from the straightforward approach to PPIL.

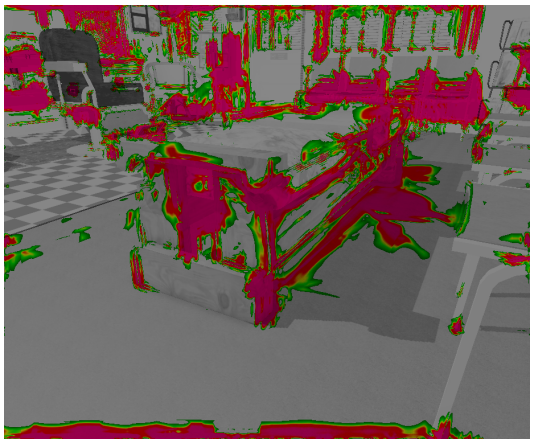
In the second configuration of the kitchen scene, a point light source is placed in the centre of the room, like with the Barbershop. Figure 7.11a and 7.11c make away with direct lighting and primary shadows and compare the indirect contribution of the straightforward approach with that of PPIL. In the straightforward approach, the constant ambient term results in a flat look, with the exception of shadows at corners or edges, due to SSAO. The indirect lighting contribution

(a) Straightforward

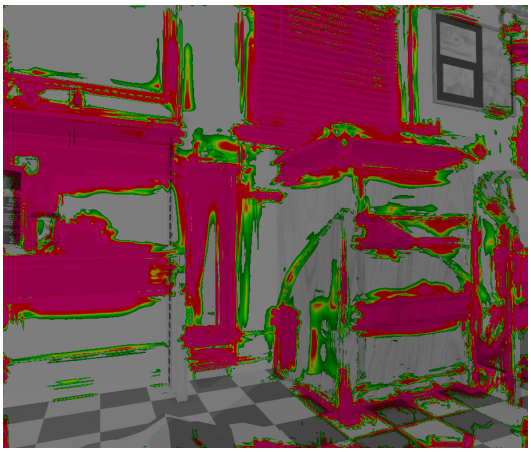
(b) Straightforward

(c) PPIL

(d) PPIL



(e) VDP



(f) VDP

Figure 7.8: Tony's Barbershop scene: scene lit by fixture inside room.

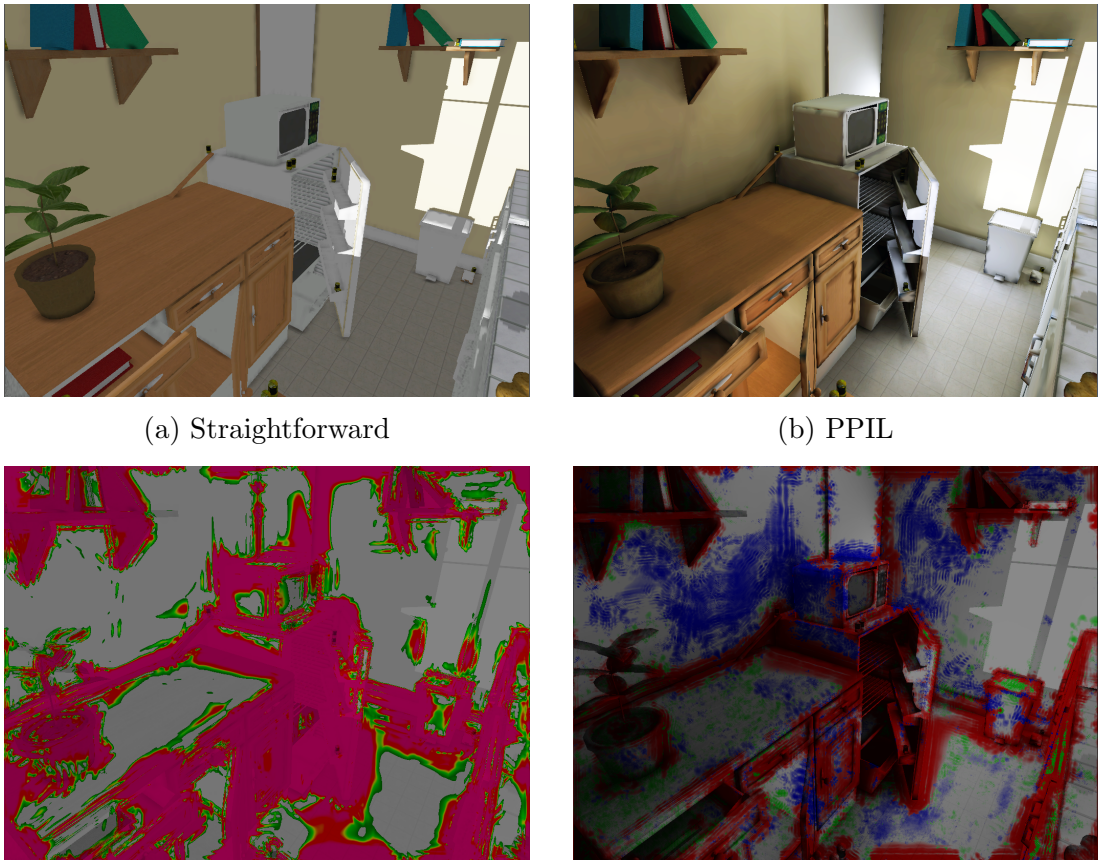
(a) Straightforward (b) PPIL

(c) Straightforward (d) PPIL

Figure 7.9: Tony's Barbershop scene: detail view of second configuration.

in PPIL is more rich; providing penumbras that are smooth and more plausible than those captured by SSAO.

Figure 7.11b and 7.11d augment the images mentioned above with direct lighting. The differences between the two images become less marked when direct lighting is added. However, for the general scene there is no guarantee that direct



(a) Straightforward (b) PPIL

(c) VDP (d) DRIM

Figure 7.10: Kitchen scene: first configuration, with light source positioned outside room.

lighting is present and to what intensity, so a GI solution is typically preferred if available. The difference images (Figure 7.11e and 7.11f) point towards occluded and unlit areas as being the major cause of disparity between the two methods.

Figure 7.12 compares shadows from secondary bounces of light generated by the two methods. The limitations of SSAO clearly show in the straightforward approach, where the shadows follow edges and corners but do not take into consideration other global features such as the open door of the appliance. The indirect shadow in the view rendered using PPIL manifests global properties, where the light bouncing on the far wall is correctly blocked by the fridge door.

Temple

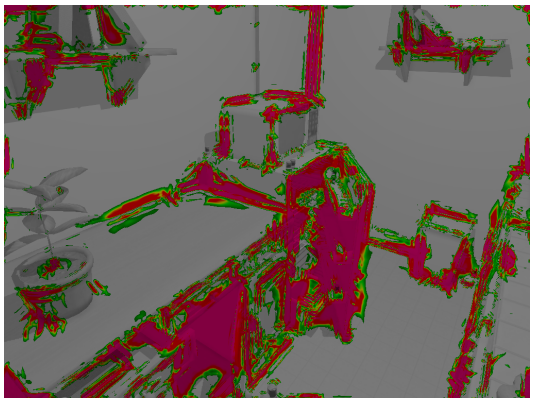
The third evaluated scene is shown in Figure 7.13. A single configuration is considered for this scene with a parallel light source used to simulate sunlight

(a) Straightforward (Ambient)

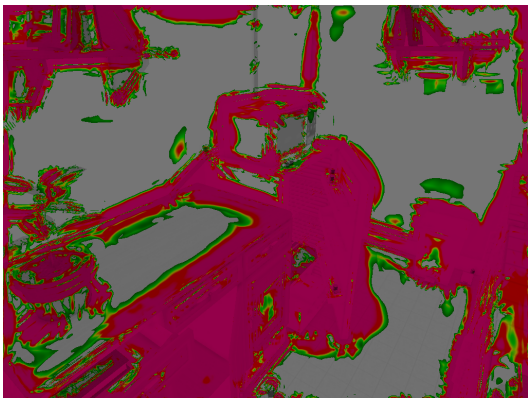
(b) Straightforward

(c) PPIL (Indirect)

(d) PPIL



(e) VDP



(f) VDP

Figure 7.11: Kitchen scene: scene lit by fixture inside room.

illumination. The particular voxel-based geometry of this scene and its well-defined edges make the ambient occlusion function feature prominently in the images rendered using the straightforward approach. In Figure 7.13a, the individual voxels at the side of the temple are more easily distinguishable than the

(a) Straightforward (b) PPIL

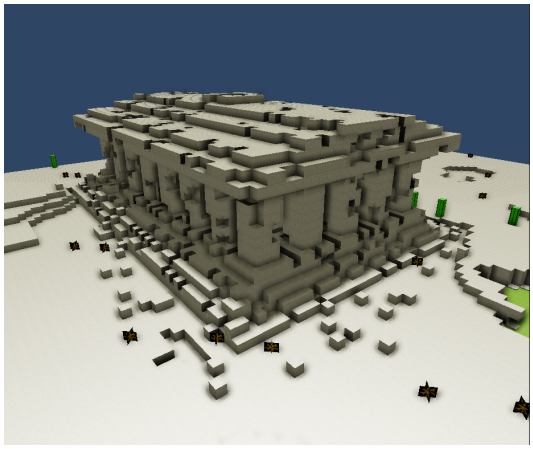
Figure 7.12: Kitchen scene: comparison of the quality of secondary shadows.

respective view rendered using PPIL and shown in Figure 7.13c. In Figure 7.13b the edges of the geometry and shape perception are enhanced by SSAO, although depth is not easily discerned. Furthermore, it could be reasonably argued that the resulting effect is neither realistic nor necessarily pleasing. The same view rendered using PPIL provides a better perception of distance and depth than the straightforward approach 7.13d.

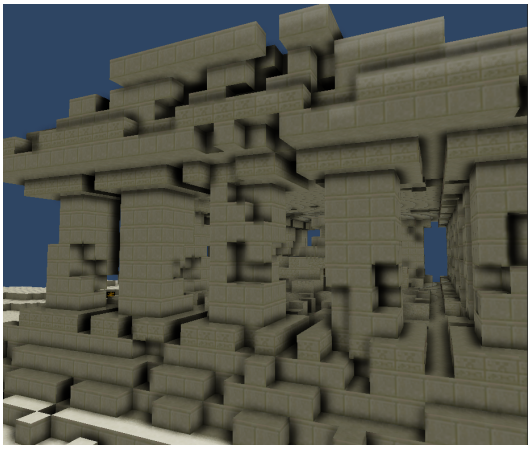
Figure 7.14a provides better visual cues where the voxels meet the green pool. The SSAO darkens the edges of the green blocks which perceptually anchors the white blocks; in Figure 7.14b, the white blocks appear to be floating over the green pool and not necessarily touching it. The straightforward approach yields a better perception of depth whereas PPIL captures diffuse interreflections. PPIL is not mutually exclusive with SSAO, and whenever a form of perceptual advantage may be gained, the two methods can be merged together as shown in Figure 7.14c. While SSAO is considered computationally cheap, it may still prove computationally prohibitive on low-end devices.

7.5 Discussion

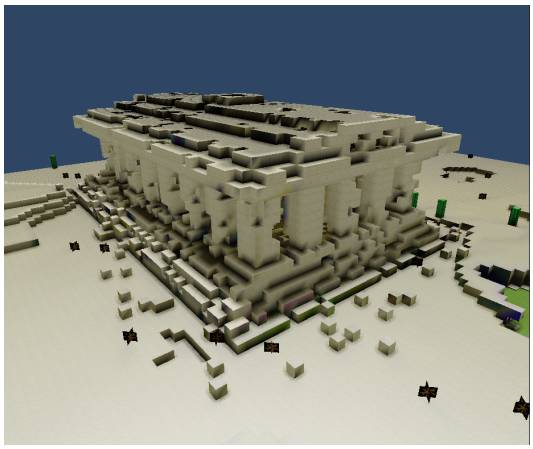
PPIL is a malleable method as it is suitable for different types of scenes. The visual quality may be adjusted to fit the computational resources available, by



(a) Straightforward

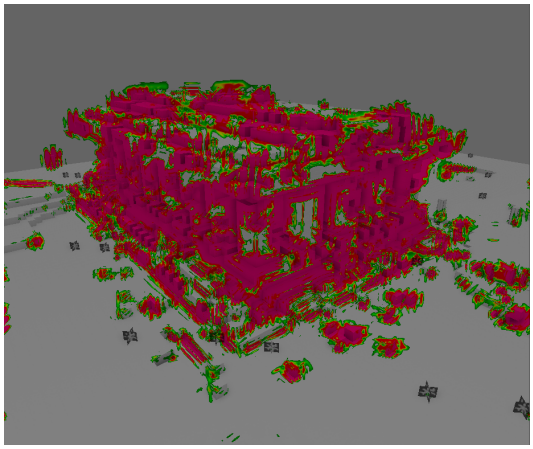


(b) Straightforward

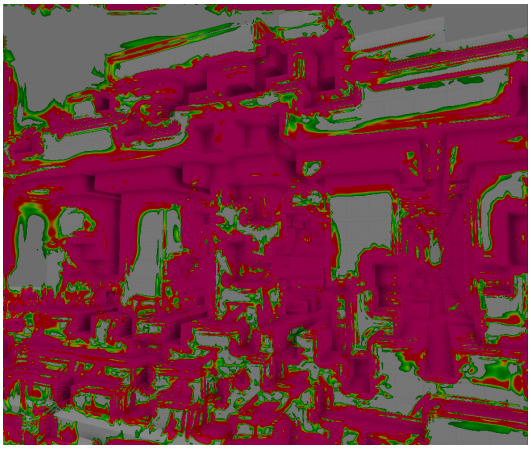


(c) PPIL

(d) PPIL

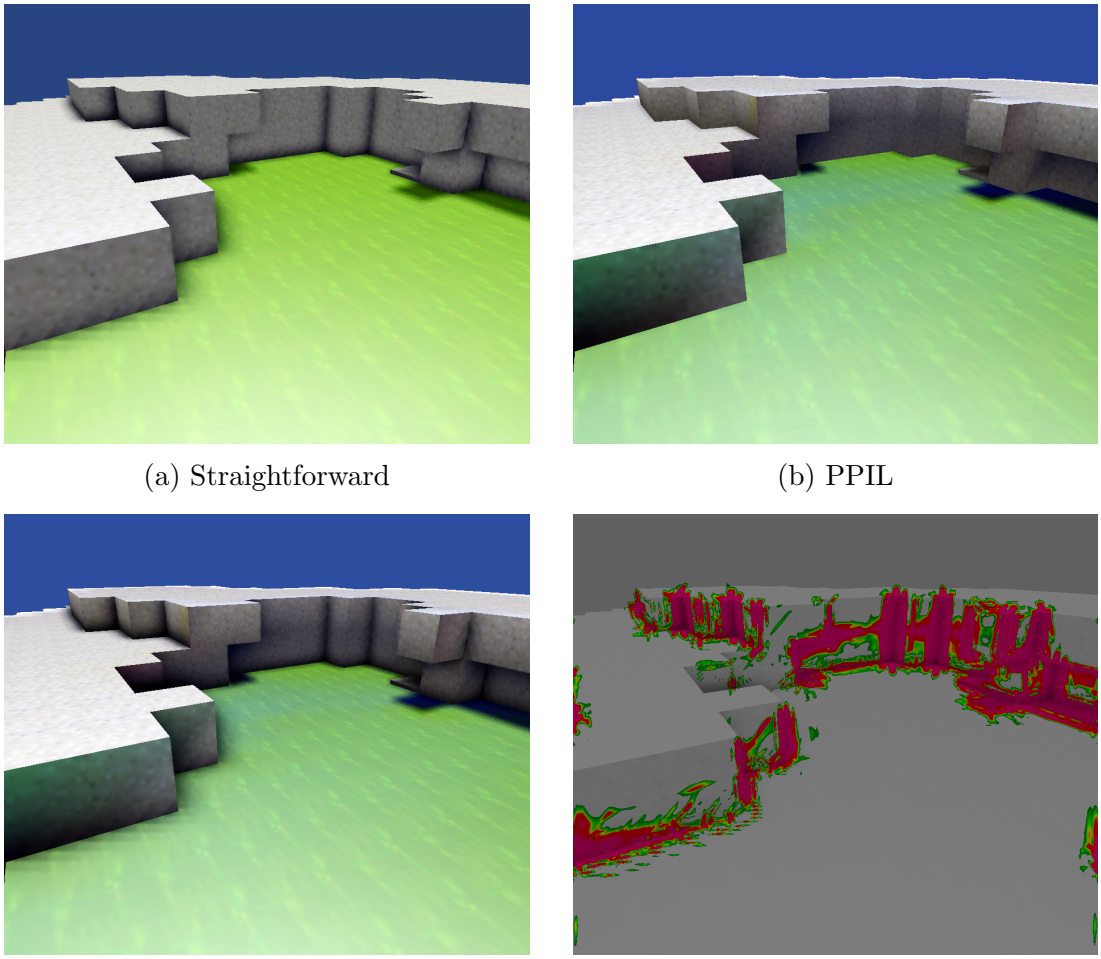


(e) VDP



(f) VDP

Figure 7.13: Temple scene: scene lit by a parallel light to simulate sunlight.



(a) Straightforward (b) PPIL

(c) PPIL + SSAO (d) VDP (Straightforward, PPIL)

Figure 7.14: Temple scene: used of SSAO to augment PPIL.

reducing the number of VPLs employed in the generation of indirect lighting, although care must be taken not to adversely affect visual quality if the parameters are set too low. Figure 7.15 illustrates how, in three difference cases, visual quality is affected by some feature of the input, be it the operating parameters or the data set acted upon. Specifically, in the left pane, artefacts typical of instant radiosity methods can be observed, where discrete shadow bands result from the use of an insufficient number of VPLs. Some scenes may prove difficult to illuminate using VPLs; in some cases, light has to travel through narrow slits or portals, such as a small window or opening in a room, and very few VPLs make it through from the light source to the scene region in question. A second class of artefacts common to instant radiosity methods is shown in the middle image. If a VPL is spatially close to a point at which the indirect lighting contribution

is being sampled, the geometry term (Keller, 1997; Debattista *et al.*, 2009) increases asymptotically and leads to bright splotches of illumination. Thus, the term is clamped to a user-provided value; the limitation, in this case, is that given the scenes are dynamically-generated, it might not always be clear as to what range this term should be clamped (see §3.5.2 for more details). A further visual inconsistency may arise where limited face tessellation results in aliasing problems. Significant changes in the low-frequency indirect lighting are missed by the sampling function and thus the extrapolation of intermediate points leads to visual inconsistencies due to missing these changes. A possible solution to this problem is the application of intelligent tessellation techniques such as discontinuity meshing (Heckbert, 1992), which has been used in similar contexts for finite element method estimation of radiosity.

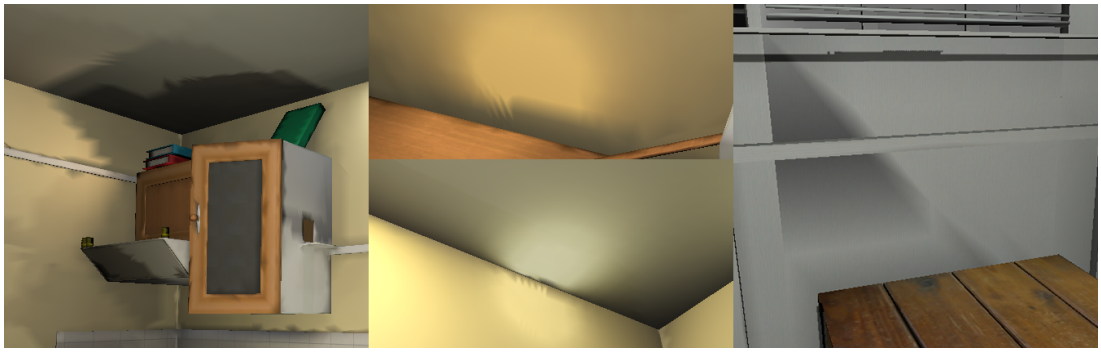


Figure 7.15: Artefacts occurring due to limitations of our method: (left) banded shadows due to insufficient number of VPLs, (middle) singularities inherent to instant radiosity methods, (right) aliasing due to limited surface tessellation.

7.6 Summary

In this chapter, the GI algorithm introduced in RAIL has been scaled down to provide precomputing indirect lighting to low-end devices. This allows the integration of computationally expensive GI effects into dynamically-generated scenes by taking advantage of the inherent connectivity available on consumer devices to distribute the cost of computing indirect illumination using physically-based algorithms, which is then stored within mesh vertices. In contrast with RAIL, and consequently RaaS, PPIL does not need a powerful central server to compute indirect illumination. It is a partially decentralised system that promotes cooperation across devices, albeit with the proviso that a precomputation step is

required and the indirect illumination doesn't change at runtime (see Table 9.3). Nonetheless, PPIL is a stepping stone towards achieving a fully decentralised system that can compute dynamic indirect lighting at runtime. This is discussed in the next chapter.

CHAPTER 8

Peer-to-peer Rendering (PePeR)

It has been shown in the previous chapters that high-fidelity rendering can be brought to a wide spectrum of devices, either through the offloading of computation to high-end servers in the cloud (see Chapters 5 and 6), or via a precomputation stage whose burden is shared by participating devices in a multi-user environment (see Chapter 7). The algorithms employed in high-fidelity rendering benefit greatly from added computational resources when exploiting parallelism, usually in the form of parallel shared memory architectures, distributed systems, vector processors or some hybridisation thereof. Many parallel rendering algorithms have been targeted at dedicated resources which are available to medium to large organisations. In contrast to these traditional client-server architectures, peer-to-peer (P2P) computing has arisen as one of the major models for offloading costs from a centralised computational entity to benefit a number of peers participating in a common activity. The P2P model provides advantages in terms of scalability, burden-sharing and fault-tolerance, while removing the need for a central authority. In the context of interactive rendering and visualisation, peers within some shared environment will invariably compute and visualise similar portions of that environment. If the result of such computations is marshalled into a global state shared across the participating peers, then these peers may be provided with results to computations they haven't yet carried out but may need to in the future. In Chapter 7, initial steps were taken in this direction where expensive computations were moved from a centralised server to the client devices themselves. Notwithstanding, the computations, the order and the workers were rigidly defined and finally executed using a bag-of-tasks paradigm. This model does not support dynamic scenes as it was never intended for such an environment. Moreover, it imposes certain constraints on the clients, such as the

participants being known a priori, agreement on an elected master via distributed consensus (unless prior knowledge is assumed), and the generation of indirect lighting as a precomputation step. Moving to a more amorphous and less rigid means of sharing computation introduces a number of challenges, especially in the light of using a shared global state to reduce redundant computations in a distributed system such as an unstructured P2P network. Specifically, it must be ensured that any changes to global shared state originating at a peer will eventually propagate through the network and cumulative operations on global shared state are correctly sequenced in order to provide some agreed upon level of consistency. This chapter introduces the concept of collaboration in high-fidelity rendering over P2P networks, with the aim of furthering the quality of the rendering by reducing redundant computation. The proposed method, PePeR (Peer to Peer Rendering), takes inspiration from the amortisation of indirect lighting computation in multi-user environments, explored in Chapters 6 and 7; the concept is generalised and adapted to the P2P paradigm.

The chapter is structured as follows: §8.1 introduces the chapter and outlines the respective contributions, §8.2 provides a specification for PePeR, discussing events and their ordering and propagation in a P2P setting, §8.3 describes the implementation of the IC algorithm in PePeR, §6.4 and §6.5 demonstrate results from PePeR followed by a discussion and §6.6 concludes the chapter.

8.1 Introduction

Large scale rendering has been mostly used in the context of offline image synthesis, with the exception of Aggarwal *et al.* (2012), whose work on desktop grids was evolved to encompass real-time rendering (see §4.5). Aggarwal *et al.* (2012), BURP (Patoli *et al.*, 2009b) and Yafrid-NG (Ramos *et al.*, 2009) operate under the volunteer computing model, where users willingly make available computing resources to be used towards solving some common problem. Both Aggarwal *et al.* (2012) and Patoli *et al.* (2009b) adopt a master-worker approach to work distribution. Ramos *et al.* (2009) adopt a more decentralised approach, where any file exchanges required during rendering are implemented using a P2P protocol. The work distribution still follows a master-worker approach, where some nodes are assigned to carrying out computation while others coordinate work distribution and gather the results. Computational resources are seen as processing

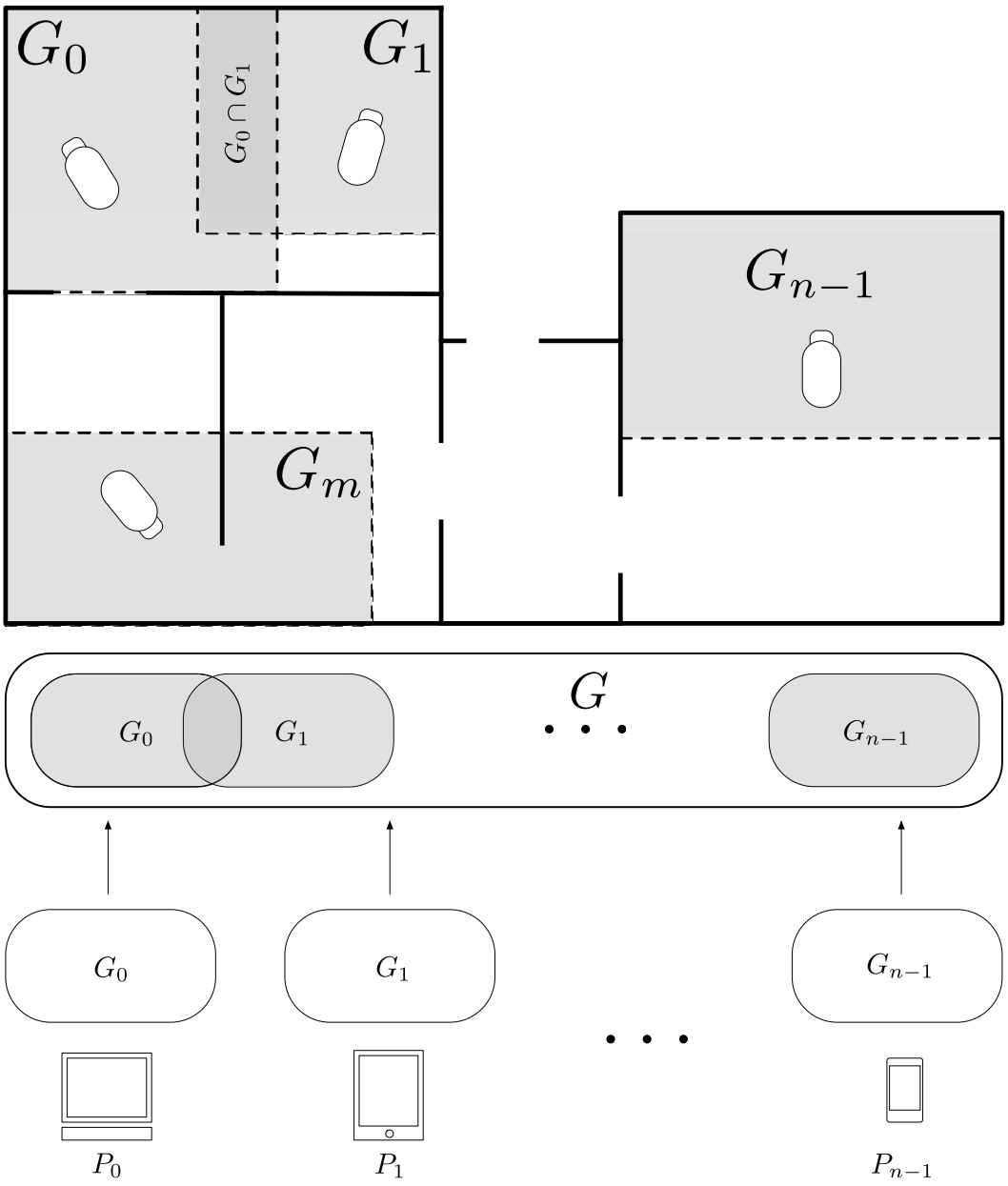


Figure 8.1: Local and global state visualisation

elements that can be assigned rendering tasks, and by which the rendering computation can be scaled to speed up the synthesis. Previous chapters focused on speeding computation using primarily centralised approaches. Converesly, PePeR shares the results of computation within a multi-user environment, to improve the overall performance of the group, where devices with lesser computational capacity are boosted by more capable members of the group. In §8.2 PePeR is discussed, starting with the introduction of the notion of global state and events.

Observable events, or events that are visible to other peers, are discussed in §8.2.1. §8.2.2 discusses event ordering in the system, following which, §8.2.3 discusses how events are propagated to peers. In §8.3, the wait-free irradiance cache (WFIC) algorithm (see §3.5.1, §4.3.1) is used in a collaborative environment as a case study, to test the viability and validity of the proposed method. The main contributions of this chapter are:

- the application of peer-to-peer to high-fidelity rendering
- the introduction of an event-based system for encapsulating ordered access to a shared data structure
- the application of an epidemiological method for event propagation within an unstructured network
- a novel collaboration algorithm for high-fidelity rendering
- a collaboration case study using the IC algorithm

8.2 Method

This section provides an overview of our method. The assumption here is that during image synthesis, each individual peer may need to carry out some computation that has already been effected by another peer, thus laying the groundwork for potential collaboration. Such computations usually entail populating data structures for caching, interpolating or generally speeding up the computation of indirect lighting. In a collaborative context, these data structures, along with others which may hold important state information describing the active scene, become part of a larger state that is shared across all the participants, P. Let $S_i = L_i \cup G_i$ be the state of participant $P_i \in P$, where L_i is P_i's internal state and G_i is the shared state.

8.2.1 Observable Events

An internal event e^{int} is the result of a write that modifies internal state L_i and potentially influences shared state. An observable event e^{obs} is the result of any significant change in internal state and is responsible for exposing a series of events $e^{int}_n, e^{int}_{n+1}, \ldots, e^{int}_{m-1}, e^{int}_m$ as part of the shared state G_i. In practice, an

observable event is abstracted to delineate and encapsulate a number of smaller, logically-related changes. For example, in an algorithm such as the irradiance cache, where irradiance samples are computed and inserted into an octree, a significant change would be represented by the insertion of an agglomeration of generated samples into the acceleration structure, as opposed to that of a single sample. An enumeration of P gives an index set $I \subset \mathbb{N}$, where $f : I \to P$ is the particular enumeration of the set of participants P. A useful abstraction we adopt is the grouping of all observable events into the global state G as represented by the shared state at each peer P_i, such that $G = \bigcup_{i \in I} G_i$. In line with our definition of an observable event as a significant change in state, we also define a mechanism by which we can determine what qualifies as a significant change in state. Specifically, for an event e_j^{int}, if the state S_i satisfies a predicate Q, then an observable event e_k^{obs} is produced and merged with the global state G_i (Fig. 8.2). The sequence of observable events $e^{obs} \in G_i$ generated by P_i is guaranteed to be ordered in time, i.e., $e_m^{obs} \to e_{m+1}^{obs}$, where $a \to b$ means event a precedes event b insofar as events are generated by the same peer. It would be desirable to extend this guarantee to events generated by other peers, establishing some form of event ordering across all collaborating peers, since observable events may represent cumulative or dependent updates to shared data structures. For example, in the case of events that invalidate the contents of a data structure, one should be able to establish whether a specific event has happened before, after or was concurrent to the invalidation.

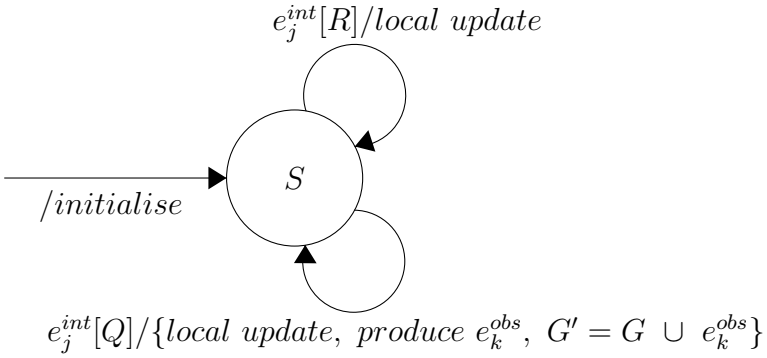


Figure 8.2: Internal event loop and observable event generation.

8.2.2 Logical Order of Events

In distributed systems, especially in decentralised applications, time-of-day clocks may be skewed or suffer from drift, and thus, global (or absolute) time ordering of events could lead to unexpected results (Krzyzanowski, n.d.). The casual order of events is captured via the use of logical clocks instead of physical ones (Lamport, 1978). A logical clock is an n-tuple $n \leq |P|$; a participant P_i is responsible for incrementing the i^{th} element of its logical clock whenever an observable event occurs (Fidge, 1988). With the help of a logical timestamp function $\mathcal{V}(e^{obs})$, we attempt to determine the system-wide ordering for an event e^{obs}; consider two events e_i^{obs} and e_j^{obs} with vector timestamps $\mathbf{V} = \mathcal{V}(e_i^{obs})$ and $\mathbf{V}' = \mathcal{V}(e_j^{obs})$ respectively:

Rule 1: e_i^{obs} happens before e_j^{obs} ($e_i^{obs} \rightarrow e_j^{obs}$) when each element of $\mathbf{V}$ is less or equal to the respective element in $\mathbf{V}'$, i.e., $e_i^{obs} \rightarrow e_j^{obs} \iff \mathbf{V}[n] \leq \mathbf{V}'[n]$ for $n \in I$ (see §8.2.1).

Rule 2: $\mathbf{V}$ and $\mathbf{V}'$ are said to be equal if their respective elements are equal, i.e. $\mathbf{V} = \mathbf{V}' \iff \mathbf{V}[n] = \mathbf{V}'[n]$ for $n \in I$.

Rule 3: Events e_i^{obs} and e_j^{obs}, with timestamps $\mathbf{V}$ and $\mathbf{V}'$ where $\exists\ n, m \in I : (\mathbf{V}[n] > \mathbf{V}'[n]) \wedge (\mathbf{V}[m] < \mathbf{V}'[m])$, are not causally related but rather denote concurrent events ($e_i^{obs} \mid\mid e_j^{obs}$).

A peer generates observable events and tags them with a logical timestamp. The logical clock of the peer is also updated to reflect the generated event (Algorithm 11). Events are then communicated to other peers via a propagation mechanism (see §8.2.3). The three rules above are used to determine the order of the propagated events before they are committed at the receiving peer. In case of conflict due to difficulty ordering concurrent events, a tie breaking function $T(e_i^{obs},\ e_j^{obs})$ is used to deterministically resolve the tie in favour of one event or the other. Each observable event generated may increase the cardinality of the global state G (since $G = \bigcup_{i \in I} G_i$), and in scenarios where event generation occurs at high frequencies, this can result in uncontrolled growth. This growth is mitigated via the use of special observable events called *grouping events* (e^{grp}), which group a set of observable events into a single event, retiring the members of the set in the process.

Algorithm 11 Updating of logical clocks

```
1: function UpdateClock(P_s, P_d)
2:     Tick(P_s)
3:     V'_{P_d} ⇐ sup(V_{P_s}, V_{P_d})
4:     Tick(P_d)
5: end function
6:
7: function Tick(P_i)
8:     Increment V_{P_i}[i] by 1
9: end function
```

8.2.3 Event Propagation

The propagation of observable events between peers employs strategies common to epidemic processes (Bailey *et al.*, 1975). Specifically, an *anti-entropy* strategy is used whereby each peer regularly chooses another peer at random (see §8.2.4) with which to exchange observable events (Algorithm 12). The aim of this exchange is that of harmonising the global state of each peer such that for peers P_i and P_j, $G_i = G_j$. A peer that has not yet seen an observable event is *susceptible* to it, while a peer that has generated or can provide the event is called *infective*. A peer that assimilates an observable event into a grouping event is known as *retired* with respect to that event. This terminology is loosely based on Kermack & McKendrick (1932).

Algorithm 12 Anti-entropy propagation algorithm

Require: $|P| \geq 1 \wedge \exists\, p \in P : active(p)$

```
1: while active(P_s) do
2:     P_d ⇐ ChoosePeer(P)
3:     ExchangePeers(P_s, P_d)                ▷ see §8.2.4
4:     ExchangeEvents(P_s, P_d)
5:     Sleep(Δt)                              ▷ delay Δt before next exchange
6: end while
```

The dynamics of event propagation are based on the *push* dynamics employed in epidemic models (Demers *et al.*, 1987). Thus, propagation may be modelled using established deterministic techniques from epidemiology literature. When an observable event is generated at a peer, propagation through the network is achieved in expected time proportional to the log of the number of peers, $n = |P|$. Specifically, for large n, the exact formula is $\log_2(n) + \ln(n) + \mathrm{O}(1)$ (Pittel, 1987). The exchange of events is a two-step process, whereby the global state of each peer

is merged with the others', if found diverging (see Algorithm 13). In particular, consider two peers P_s and P_d, where P_s is the originator of the exchange and P_d the recipient. If their respective shared state, G_s and G_d, are found differing, G_s is merged with G_d at the recipient, while G_d is merged with G_s at the originator, in that order. During each merge process, the events from both states are first combined and ordered. The newly acquired events are then committed in their perceived order of occurrence. During event exchange, the logical clocks of the respective peers are updated and made consistent (Algorithm 11).

Algorithm 13 Global state synchronisation algorithm

```
1: function ExchangeEvents(P_1, P_2)
2:     if G_1 ≠ G_2 then
3:         UpdateClock(P_1, P_2)
4:         MergeEvents(G_1, G_2)
5:         UpdateClock(P_2, P_1)
6:         MergeEvents(G_2, G_1)
7:     end if
8: end function
9:
10: function MergeEvents(G_s, G_d)
11:     C ⇐ G_s ∪ G_d
12:     Commit(C, R) where R = {x, y ∈ C · Pre(x, y)}
13: end function
14:
15: function Pre(e_1,e_2)
16:     if e_1 = e_2 then
17:         return false
18:     end if
19:     if V(e_1) = V(e_2) then
20:         return T(e_1, e_2)
21:     else if V(e_1) ≤ V(e_2) then
22:         return true
23:     else if V(e_2) ≤ V(e_1) then
24:         return true
25:     else
26:         return T(e_1, e_2)
27:     end if
28: end function
```

An example of event propagation, ordering and merging is shown in Figure 8.3. In this example, four peers (P_1 through P_4) participate in the network. Three observable events (a, b, c) are generated by P_1, P_2 and P_4 respectively. The

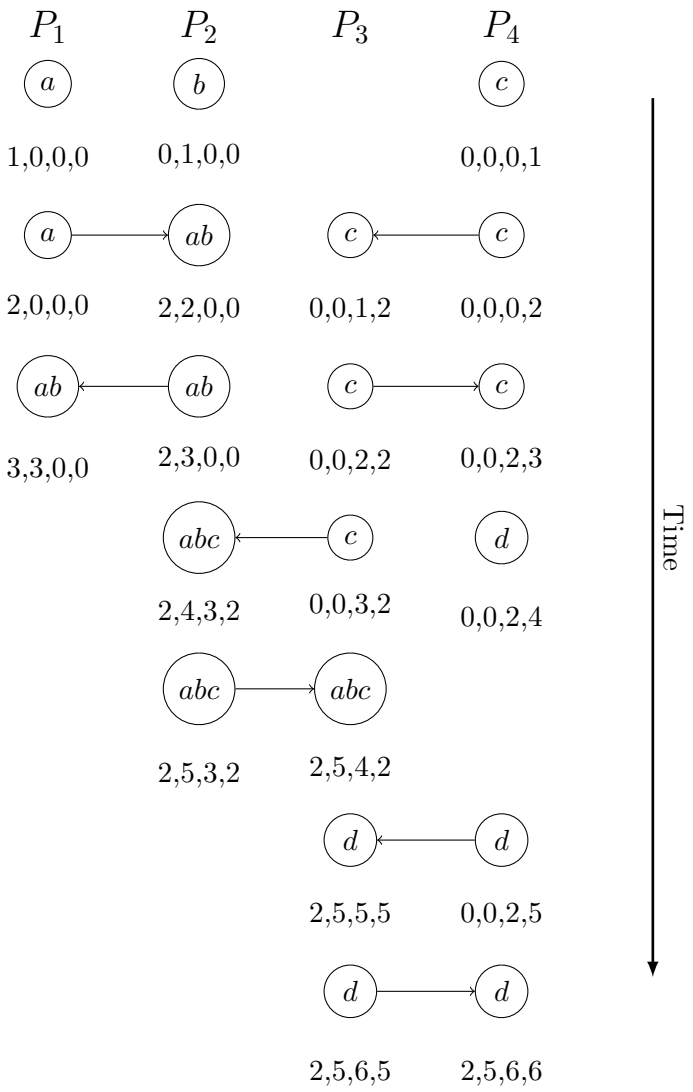


Figure 8.3: Example of event propagation, ordering and merging of global states.

resulting timestamps can be observed below the events for the respective peers. Subsequently, an exchange ensues between P_1 and P_2, and another between P_3 and P_4. In the first case, events a and b are concurrent (**Rule 3**) and thus resort to a tie-breaking function which orders events by process id, before merging and resulting in $G_1 = G_2 = \{ab\}$. In the second exchange, between P_3 and P_4, P_3 simply acquires the event, such that $G_3 = G_4 = \{c\}$. Next, P_1 leaves the network, P_2 and P_3 start an exchange, while P_4 produces an invalidation event d. The exchange between P_2 and P_3 results in $G_2 = G_3 = \{abc\}$, while the invalidation event d at P_4 results in c being removed. In the next stage, P_2

leaves the network and a final exchange ensues between P_3 and P_4. The ordering process promotes d as the most recent event by virtue of it being an invalidation event; the tie-breaking function between an invalidation event and a standard event will always break the tie in favour of the standard event (i.e., it is assumed that the standard event happened before the invalidation event). In the case the an irradiance cache, a standard event may be thought of as the agglomeration of a number of irradiance samples computed and inserted into an octree. The invalidation event could be any event that invalidates these samples, like turning a light source on and off which was used in generating these samples (see §8.3).

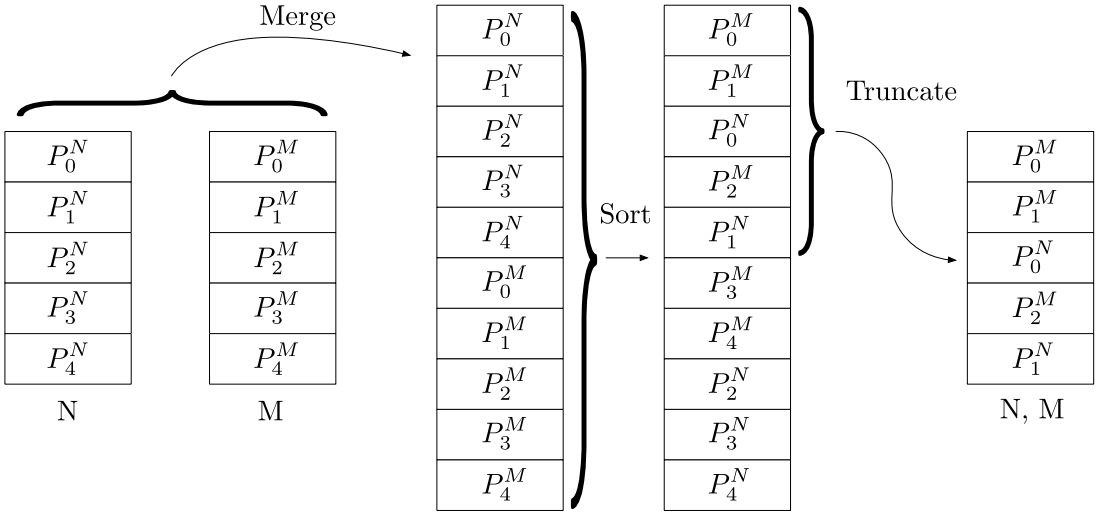


Figure 8.4: Peer discovery mechanism

8.2.4 Peer Discovery and Membership

Unstructured P2P systems aim at exploiting randomness to disseminate information across a large set of nodes (Voulgaris *et al.*, 2005). Thus, membership, or the way a collaborating peer learns about other peers, is fundamental because it controls the performance of subsequent disseminations. Connections between nodes in gossiping networks are highly dynamic and often need to obtain random samples from the entire network in order to periodically exchange information with random peers. Membership is handled in a number of different ways; a straight forward approach is that of furnishing the peers with a fixed directory provider, which lists all collaborating peers in the network. This approach requires maintaining global information which may prove to be problematic, especially in the

case of major network disasters (Voulgaris *et al.*, 2005). The approach taken is similar to the newscast method (Jelasity & van Steen, 2002), where it was elected to keep a finite cache of peers instead of all the members of the network. Each peer is tagged with a logical timestamp representing the last communication event associated with it. The cache needs to be cold started by populating it with at least one peer which is already a member of the network. Subsequently, at each exchange caused by the anti-entropy algorithm (Algorithm 12), the caches of the two peers taking part in the exchange are merged, possibly resulting in a number of peers twice the size of each individual cache. The eviction policy used is conceptually similar to a *least recently used* strategy, where the list of peers is sorted by their logical timestamp and the top k entries are retained, where k is the size of an individual cache.

8.3 Irradiance Caching over P2P

The specification presented in §8.2 has been envisaged to be as generic and flexible as possible, supporting the integration of a number of high-fidelity techniques that are view-independent and can compute progressive solutions on demand. For example, radiosity methods (§3.6), albeit being view-independent, do not satisfy the requirement for on demand computation. Photon Mapping (Jensen, 2001) is also view-independent but requires a precomputation step that, similarly to radiosity methods, violates the on demand requirement. Progressive Photon Mapping (Hachisuka *et al.*, 2008), which uses ray tracing as a first pass and a number of photon tracing passes subsequently, could share a photon map built in each of the photon tracing passes, satisfying both view independence and progressive solution computation. The Irradiance Cache (Ward *et al.*, 1988) is another method that is amenable to integration into the P2P framework, and although it generates view-independent information, the demand is driven by what the observer's view requires. The Irradiance Cache (IC) is an object-space data structure that caches the computation of diffuse indirect lighting at a number of points sparsely sampled over surfaces in a scene (see §3.5.1). The records stored in the IC can be reused for rendering multiple frames with different viewing parameters, provided the scene configuration, in terms of geometry or lighting, doesn't change. In multi-user environments, this can be especially advantageous since participants can compute indirect lighting once and share it, potentially

cutting down on the computation time required to generate a global illumination solution (see Figure 8.5).

Figure 8.5: Collaboration using the irradiance cache

In both shared and distributed memory paradigms, efficient concurrent access to the IC is a challenging task (Debattista *et al.*, 2011). In a peer-to-peer setting, this is exacerbated by the lack of a centralised authority coordinating access to the data structure. Moreover, providing a consistent view of the IC to the various participants while consolidating updates presents an additional challenge. At a high level, operations on the IC are similar to a traditional cache: read, write and invalidate. Read operations involve traversing the data structure, which is predominantly an octree, to search for valid irradiance records, while write operations, besides traversing the tree, also add nodes and records to it. Invalidation is the process by which records stored in the IC are tagged as out of date, or corrupt. For instance, when light sources or objects in the environment move, the records in the IC that are affected by the resulting change in the lighting become corrupt and in need of re-computation; they are thus invalidated to prevent

their use. There are methods for keeping track of the individual records that have been affected by some change in scene configuration as shown by Debattista *et al.* (2009), although in this work, a straightforward approach has been taken where all records in the data structure are marked during an invalidation operation.

8.3.1 Observable Event Generation

In line with the model of observable events, two predicates have been identified to capture the behaviour governing write access (insert and invalidate) to the shared data structure. The predicates, Q_{ins} and Q_{inv}, when satisfied, generate the observable events E_{ins} and E_{inv} respectively. The event E_{ins} inserts a batch of irradiance records of size $recordsPerEpoch$ into the IC, while E_{inv} invalidates the data structure and all records contained therein, unconditionally clearing the IC. The predicate Q_{ins} is defined as follows:

$$Q_{ins} = recordsInserted \geq recordsPerEpoch \tag{8.1}$$

where $recordsInserted$ is a tally of the number of irradiance samples that have been inserted into the IC. This value is initially set to zero and is reset every time an observable event of type E_{ins} is generated. In practice, this could be seen as an instance of the producer-consumer pattern, where $recordsInserted$ is the number of used buffers, and $recordsPerEpoch$ is the total number of buffers. As soon as all the buffers are full ($recordsInserted \geq recordsPerEpoch$), an event is generated that copies the contents to some other memory and empties the buffers. The *epoch*, which is discussed further on, is used to uniquely identify an agglomeration of $recordsPerEpoch$ samples inserted into the IC on a local machine (i.e., *epoch* values are not unique across different machines). The second predicate, Q_{inv}, is defined below:

$$\begin{aligned} Q_{inv} = \exists\ x \in (objects \cup lights), y \in lights\ \cdot \\ trasformChanged(x)\ \vee\ propertiesChanged(y). \end{aligned} \tag{8.2}$$

The position, scale and orientation of each dynamic object is represented by a transform $T \in R^{4\times4}$; $transformChanged$ determines whether T has changed between frames $t-1$ and t, i.e. $\delta(T_{t-1} - T_t) = 0$. Similarly, any change in properties associated with light sources, such as power or spotlight cone angle,

are captured by *propertiesChanged* as shown below:

$$propertiesChanged(x) = \exists y \in properties(x) \cdot [\delta(y(t) - y(t-1)) = 0], \quad (8.3)$$

where $properties(x)$ enumerates the set of properties associated with light source x. For every observable event that results from satisfying either of these two predicates (equations 8.1, 8.2), a Universally Unique Identifier (UUID) is generated to uniquely address the event and differentiate it from every other observable event generated in the system (network). The event is also timestamped using a vector clock to help establish a system-wide ordering of events (see Algorithm 14).

Algorithm 14 Generation of observable event

```
1: function RecordEvent_ins(P_1)
2:     if Q_ins then
3:         eventID ⇐ uuid.Next
4:         epochID ⇐ epoch.Next
5:         time ⇐ clock.Tick
6:         RecordEvent(time, eventID, epochID, Insert)
7:     end if
8: end function
9:
10: function RecordEvent_inv(P_1)
11:     if Q_inv then
12:         eventID ⇐ uuid.Next
13:         time ⇐ clock.Tick
14:         RecordEvent(time, eventID, null, Clear)
15:     end if
16: end function
```

8.3.2 Observable Event Merging

Generated observable events are propagated across the network to the other peers. The mechanism by which propagation occurs demands exchanges between pairs of peers, during which an anti-entropy phase is carried out to ensure consistency of global state (see §8.2.3). During this phase, observable events from both peers are ordered and merged into the respective peers' local state. Algorithm 13 (lines 11-12) illustrates how, the method by which the two sets of observable events are grouped and ordered, is independent of the underlying collaboration structures

(the IC in this case) and only requires the tie-breaking functions between different event types to be defined.

On the other hand, the merging of observable events is highly specific to the nature of the collaboration and the minutiae surrounding access to the data structures being shared. In the specific case of the IC, two event types have been identified which effect some change to the shared state, E_{ins} and E_{inv} (§8.3.1). The insertion of individual records in the IC is unaffected by the insertion order. Thus, insertion events (E_{ins}) are order-independent provided that no invalidation event (E_{inv}) occurs in the event trace. More specifically, given a two-event trace, $e_m^{obs}, e_n^{obs} \in E_{ins}$, merging can occur in any order, and furthermore, the events in the trace can be conveniently composed into a single event $e_k^{obs} = e_m^{obs} \circ e_n^{obs}$, where $\circ$ is the composition operator and $e_m^{obs} \circ e_n^{obs} = e_n^{obs} \circ e_m^{obs}$ where $e_m^{obs}, e_n^{obs} \in E_{ins}$.

In practice, as long as the two insertion events occur contiguously, the associated irradiance records may be inserted in the IC in any order. This holds even for concurrent observable events where no causal relationship exists. The system specification still expects a total ordering of the events, and since the composition operator is commutative, establishing such an ordering reduces to providing a tie-breaking function that evaluates deterministically across the network. Inspired by Lamport (1974), the UUID of an event is used as a priority value such that $T(e_m^{obs}, e_n^{obs}) = UUID_m < UUID_n$, with the total ordering of the events given by:

$$e_m^{obs} \Rightarrow e_n^{obs} = (e_m^{obs} \rightarrow e_n^{obs}) \vee \left((e_m^{obs} \parallel e_n^{obs}) \wedge T(e_m^{obs}, e_n^{obs})\right). \tag{8.4}$$

A number of established algorithms for generating UUIDs guarantee either that the generated value is unique or extremely likely to be unique given the low probability of generating two duplicates. A relation between the UUIDs of two events can be established, similarly to an inequality, that deterministically orders them irrespective of anything but their actual values.

Invalidation events can be likewise ordered and merged. These events are conceptually simpler than insertion events in that they reset the IC to its initial state, before any records had been inserted. Intuitively, in a trace of contiguous invalidation events, only the first event performs any meaningful change to the state of the IC; subsequent invalidations are redundant since no records have been added since the last invalidation. Thus, such an event trace can be collapsed into a single invalidation event. Assuming total ordering of E_{inv} type events can be

established using Equation 8.4, the composition operator for the respective event type is defined as:

$$e_m^{obs} \circ e_n^{obs} = e_n^{obs} \circ e_m^{obs} = \begin{cases} e_n^{obs} & e_m^{obs} \Rightarrow e_n^{obs} \\ e_m^{obs} & otherwise \end{cases}, \tag{8.5}$$

where $e_m^{obs}, e_m^{obs} \in E_{inv}$. So far, the composition operator has been defined for operands of the same type, mapping $E_{ins} \times E_{ins}$ to E_{ins} and $E_{inv} \times E_{inv}$ to E_{inv}, but there are instances where two events of different types can be composed into a single one and it is convenient to do so. In the case of an invalidation following an insertion event, it is clear that independent of the changes carried out by the insertion event to the IC, the invalidation will reset the structure and remove all records. Thus, the insertion event can be discarded and the two events composed into a single invalidation event $e_m^{obs} \circ e_n^{obs} = e_n^{obs} \circ e_m^{obs} = e_n^{obs}$, where $e_m^{obs} \Rightarrow e_n^{obs}$ and $e_m^{obs} \in E_{ins},\ e_n^{obs} \in E_{inv}$. If the order of events is inverted, where the invalidation precedes the insertion, there cannot be a reduction to a single event of either type except in the special case where the IC is empty and the invalidation becomes redundant. Thus for events $e_m^{obs} \in E_{ins},\ e_n^{obs} \in E_{inv}$ where $e_n^{obs} \Rightarrow e_m^{obs}$, composition is undefined.

If the ordered event list resulting from an exchange between two peers is treated as a string of totally ordered event types, composition may be used to reduce the string to a more compact but semantically equivalent one. From the definition of observable event composition semantics above, a number of inference rules can be constructed to minimise such strings:

Rule C1: $\dfrac{E_{ins}, E_{ins}}{E_{ins}}$ (using composition of insertion events)

Rule C2: $\dfrac{E_{inv}, E_{inv}}{E_{inv}}$ (using composition of invalidation events)

Rule C3: $\dfrac{E_{ins}, E_{inv}}{E_{inv}}$ (using composition of insertion followed by invalidation)

A string is parsed right to left, and for each pair of event types, the appropriate rule is applied. When no rule exists to reduce an encountered pair, the right symbol is emitted as is, and a new pair is formed by parsing the next symbol in the string. For example, consider the string of totally ordered event types [*aaaabbbaabaa*], where $a = E_{ins}$ and $b = E_{inv}$; this string can be minimised to the equivalent string [*ba*], as shown in Figure 8.6 using the rules above. Starting

from the rightmost symbol, the first pair is formed (a, a). Applying **Rule C1** yields a. The next pair is (a, b), and applying **Rule C3** gives b. The subsequent pair (b, a) cannot be reduced and thus, a is emitted as is, while a new pair (b, b) is formed by using the next symbol.

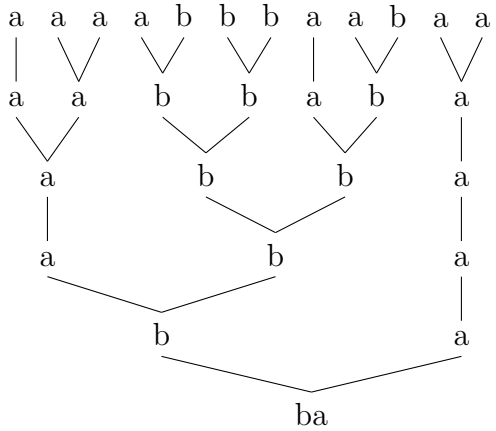


Figure 8.6: Minimisation of a totally-ordered event type string.

When the leftmost symbol has been processed, a new string [*aabbaba*] is generated, which can be reduced further. In particular, **Rule C1** or **Rule C2** can repeatedly be applied to contiguous clusters of similar symbols until each cluster has been reduced to a single symbol. For example, [*aaaaabbbbbaaabbabaabaaa*] can be reduced to [*ababababa*]. Using **Rule C3**, [*ab*] substrings can be reduced to [*b*], giving [*bbbba*] for the previous example, which further reduces to [*ba*] (**Rule C2**). The outcomes of all possible reductions are [*a*], [*b*] or [*ba*], for strings generated by the expressions a^*, $(a|b)^*b$ and $(a|b)^*b\ a^*$ respectively, and thus reduction is iteratively applied until the resulting string matches one of the three base cases above. Since every event trace reduces to one of these base cases where, if present, b is a prefix, merging with the local state of a peer need not proceed further than the first invalidation event encountered, parsing the trace from the most recent to the least recent event.

8.3.3 Tie-breaking Functions

Reiterating from the previous discourse, tie-breaking functions are important because they bring total order to an otherwise partially-ordered system of events. This is important because in a partially-ordered set of events, it is not the case that for any two events, one will always strictly precede the other ($e_m^{obs} \rightarrow e_n^{obs}$

or $e_n^{obs} \rightarrow e_m^{obs}$ for all e_n^{obs}, e_m^{obs}), since it is possible for the events to be concurrent ($e_m^{obs} \parallel e_n^{obs}$). Thus, for any two concurrent events, two different peers may come to different conclusions when determining which event should take priority over which. Total ordering of the event set, on the other hand, guarantees that across the network, and on all different peers, for any two events, it can always be ascertained which event precedes the other ($e_m^{obs} \Rightarrow e_n^{obs}$ or $e_n^{obs} \Rightarrow e_m^{obs}$ for all e_n^{obs}, e_m^{obs}). The tie-breaking functions augment the partial-ordering operator $\rightarrow$, to provide this total ordering (see Equation 8.4).

Tie-breaking functions between similarly-typed events use their UUID to break the tie in favour of one or the other (§8.3.2). Tie-breaking across the two different event types identified in the IC case study (§8.3.1) is accomplished by assigning priorities to the types themselves. More specifically, insertion events (E_{ins}) are always given higher priority with respect to invalidation events (E_{inv}). In the context event order, this means that the insertion event precedes the invalidation event. In terms of tie-breaking functions:

$$T(e_m^{obj},\ e_n^{obj}) \cdot \forall e_m^{obj} \in E_{ins}, e_n^{obj} \in E_{inv} \tag{8.6}$$

$$\neg T(e_m^{obj},\ e_n^{obj}) \cdot \forall e_m^{obj} \in E_{inv}, e_n^{obj} \in E_{ins}, \tag{8.7}$$

the tie will be always broken in favour of the insertion event, such that it will happen before the invalidation. Prioritising insertions is advantageous because the resulting event type substring can be reduced using **Rule C3** above.

8.3.4 Event Grouping

Events are disseminated across the network via peer exchanges; the higher the number of exchanges from the generation of an event, the less likely it is to find a peer that has yet to learn about it. Therefore, during these exchanges, longer lived events are less likely to contribute to the exchange, making their broadcasting, as time goes by, redundant. Logically grouping these events into some aggregation could make exchanges more efficient. The premise of grouping events is based on rumour spreading, which is also founded in epidemic theory. In particular, when two peers try to exchange an event both have, the event has a probability $\frac{1}{k}$ of being retired into a group. This reflects the idea of a person (infective) who tries to spread a rumour within a group of n other persons (susceptible) by randomly calling people in the group, one at a time. When a

call results in the other person already knowing the rumour, the caller starts losing interest in actively spreading the rumour (Demers *et al.*, 1987). In PePeR, k is set to the size of the cache of peers held at each client (see §8.2.4). Event grouping is aimed at reducing the amount of information exchanged by peers, where group tags are broadcast instead of the individual event identifiers. These tags are generated via hashing of the UUIDs of the events contained in the group, which only includes insertion events.

8.3.5 Wait-free Irradiance Cache

The exchange of events for dissemination across the network executes in parallel with the rendering, to avoid creating potential bottlenecks when large exchanges are being carried out. Both processes can concurrently read and write to the IC, which may lead to race conditions, unless access to the data structure is properly controlled. Debattista *et al.* (2011) have shown that a lock-based approach to IC access can impinge on performance due to contention increasing with concurrency, and proposed a wait-free algorithm for shared memory multiprocessors, the wait-free irradiance cache (WFIC). The WFIC was originally designed around a single-reference octree which has the disadvantage of a relatively slow search procedure due to the possible use of recursion in the examination of multiple children at each node (Krivanek & Gautron, 2009). Furthermore, irradiance records are stored using a single dynamic array at each node that is extended on demand. An additional structure provides wait-free access control to these arrays. This kind of record aggregation, while providing a spatially coherent view of the data, cannot provide efficient access to records aggregated using other criteria without significant bookkeeping overheads. For instance, the exchange of observable events requires accessing records in terms of their chronological insertion order, which would require keeping track of the various records spread about the different dynamic arrays at each node (see Figure 8.7).

Since both views are indispensable to the correct operation of the system, the WFIC was extended to support both. The straightforward approach was to create a list where each element chronologically maps to a record in the dynamic arrays in the octree nodes, which also introduces an indirection when accessing event-related records. For brevity and clarity, this list is referred to as `CRList`. Reversing the indirection, it was possible to store the records, also in chronological order, but link to them from the dynamic arrays instead of the other way round.

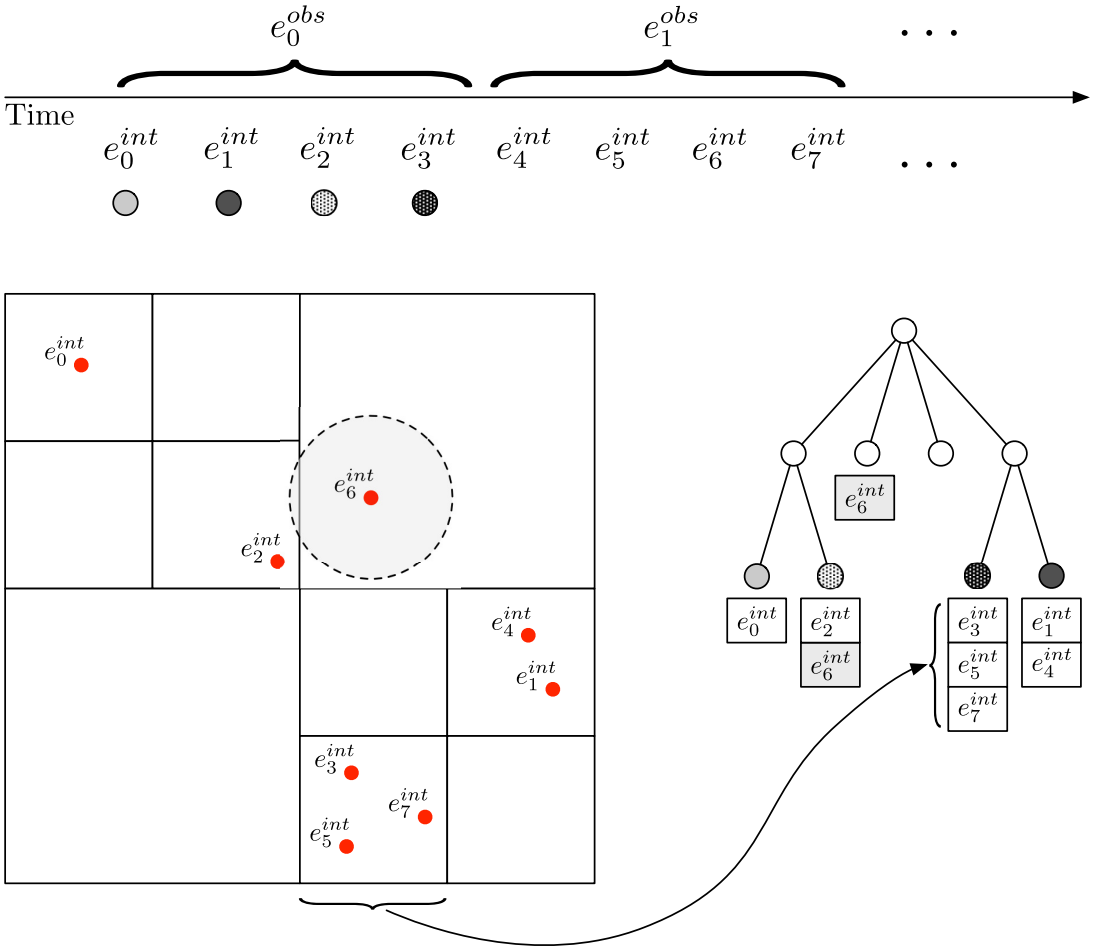


Figure 8.7: Bookkeeping of event-related records

This change permits the IC to reference the same record multiple times, and thus, at the cost of changing the traversal and insertion logic, the speed of the search could be improved by changing the associated octree to a multiple-reference one.

The `CRList` will be concurrently read and written to by multiple threads. Ensuring the correctness of the multiple reference WFIC entails showing that the list itself remains consistent through direct concurrent updates, and that at a higher level, when searching for or inserting a record into the octree, the dynamic array at each node will be consistent with it. Since the function of the `CRList` is identical to the dynamic arrays stored in the nodes of the original WFIC, then it can be shown that multiple threads can concurrently access the structure without generating race conditions. The dynamic arrays at the nodes `SPList` are also similar to the structures in the original WFIC, only differing on the type of the record data stored in the array. The original WFIC did not require writing a

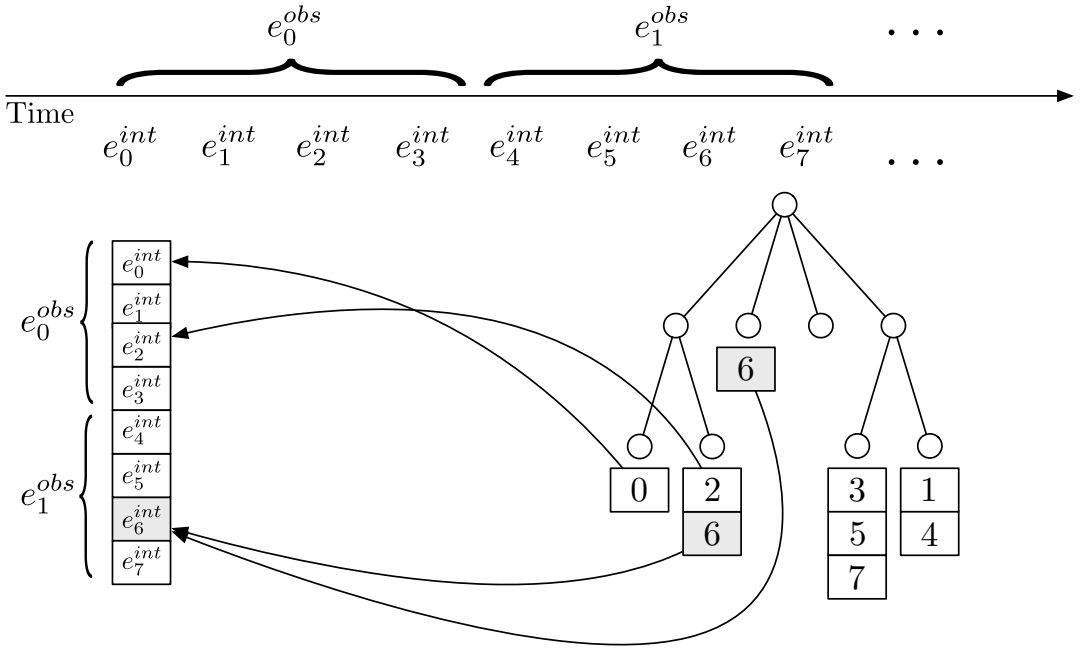


Figure 8.8: Multiple reference irradiance cache

datum to a new slot in the list to be an atomic operation. Therefore, changing the type of record in the list, without structurally altering any insertion logic, will not affect its atomicity with respect to multiple concurrent threads accessing it. It bears noting that the multiple reference WFIC, besides imposing an indirection during traversal, may also introduce more contention than the original WFIC, due to the centralised `CRList` and its position counter which is atomically incremented using the *fetch-and-add* operation. Every generated agglomeration of irradiance samples is tagged with an epoch number. The epoch number is a monotonically increasing integer which serves to identify the samples bound to a specific insertion event. There are two classes of epoch numbers, termed low epochs and high epochs. Low epochs are assigned to samples generated locally, while high epochs are assigned to those generated over the network and merged at a peer. This is immediately indicative of the fact that epochs are not unique across all the network, but are unique only at a local level. Figure 8.9 illustrates how epochs are associated with observable events that perform insertions; χ denotes the smallest high epoch value.

The merging of concurrent insertion events leads to another important performance consideration. By their very nature, concurrent events are generated on different peers. It is plausible to assume that concurrent insertion events could have been generated by peers which have similar view parameters. When merged,

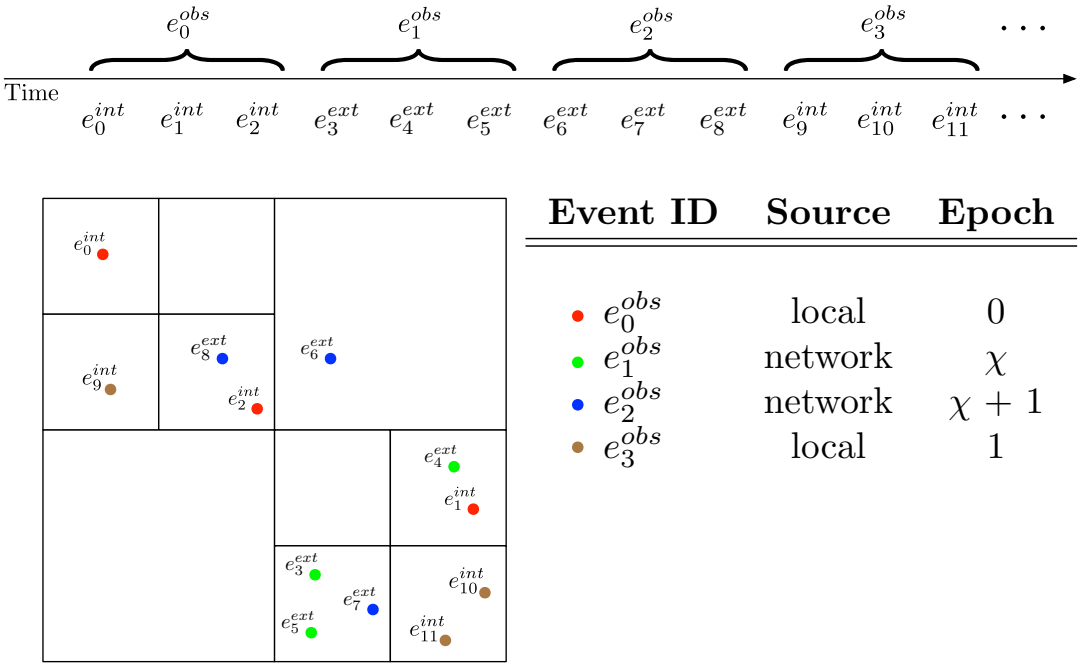
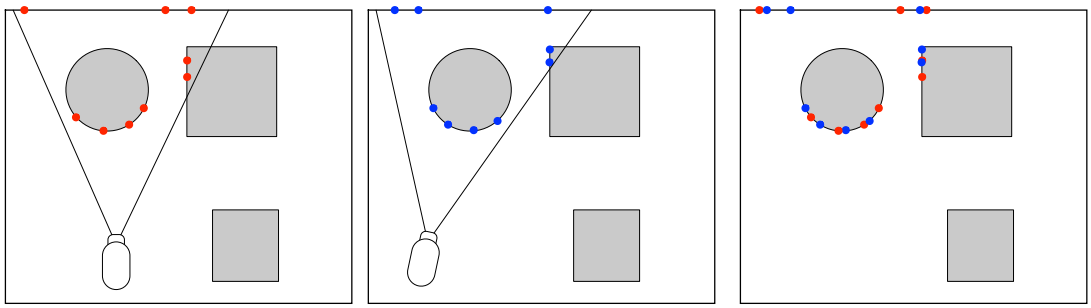


Figure 8.9: Epoch tagging of records in the IC

such concurrent events could lead to dense clusters of samples forming that would otherwise not have been independently generated, resulting in an over-saturation of the IC (see Figure 8.10). An oversaturated IC negatively impacts the interpolation of indirect diffuse lighting during rendering, slowing it down. The additional records do not necessarily contribute to a better quality image, but instead violate the density constraints of the irradiance error threshold set by the user. In order to reduce clustering and oversaturation, irradiance records generated on other peers are inserted only if they satisfy a further condition, that no samples exist with a given radius. The underlying implication when discarding irradiance records is that the structure of the IC at different peers will not be consistent, unlike the list of observable events.

8.4 Results

The specification for collaboration in a P2P environment introduced in §8.2 has been evaluated using the case study presented in §8.3. The test setup consists of a local network of eight heterogeneous machines, each equipped with 8GB of memory and quad-core multiprocessors. The data sets used in the tests have been purposefully borrowed from video games with a major online multiplayer



(a) View 1 (b) View 2 (c) Combined

Figure 8.10: Merging insertion events may lead to over-saturation of the IC. If the views are similar, aggregating samples may result in an overly crowded irradiance cache which degrades traversal performance of the data structure.

component, where the aspect of collaborative rendering can be best put to use. The use of these data sets coupled with a heterogeneous mixture of machines aim to improve the ecological validity of the study. The data sets, Town (Halflife community) and Sanctum (Quake 3 Team Arena), are shown in figures 8.11 and 8.12 respectively. A number of general parameters have been established and used for both scenes in all tests. The IC error value (α) is set to 0.15, and a total of 1.5K rays are traced to compute each new irradiance sample. Each machine runs a single peer, and is configured to attempt an exchange with another peer not earlier than $2.5s$ since the completion of the last successful exchange. For a given peer, this means that the time elapsed between one exchange and the next is at least $2.5s$. A peer will query the incoming exchange request buffer every $250ms$. Peer interaction in the virtual environment is simulated by having each perform a random walk through the scenes. In particular, for each of the two data sets considered, interest points were drawn up, and from these, a number of paths of variable length were randomly generated. A path does not necessarily span all the points of interest and may loop multiple times. Each machine was then seeded with a path, which was subsequently used across all experiments carried out. The walkthrough paths for both scenes are shown in Figure 8.13 and Figure 8.14.

8.4.1 Speed-up in Quiescent Networks

In order to contextualise any possible gains due to collaboration, the speed-up in a quiescent network is first measured. We define a quiescent network as one where no more updates are being carried out and observable events have prop-

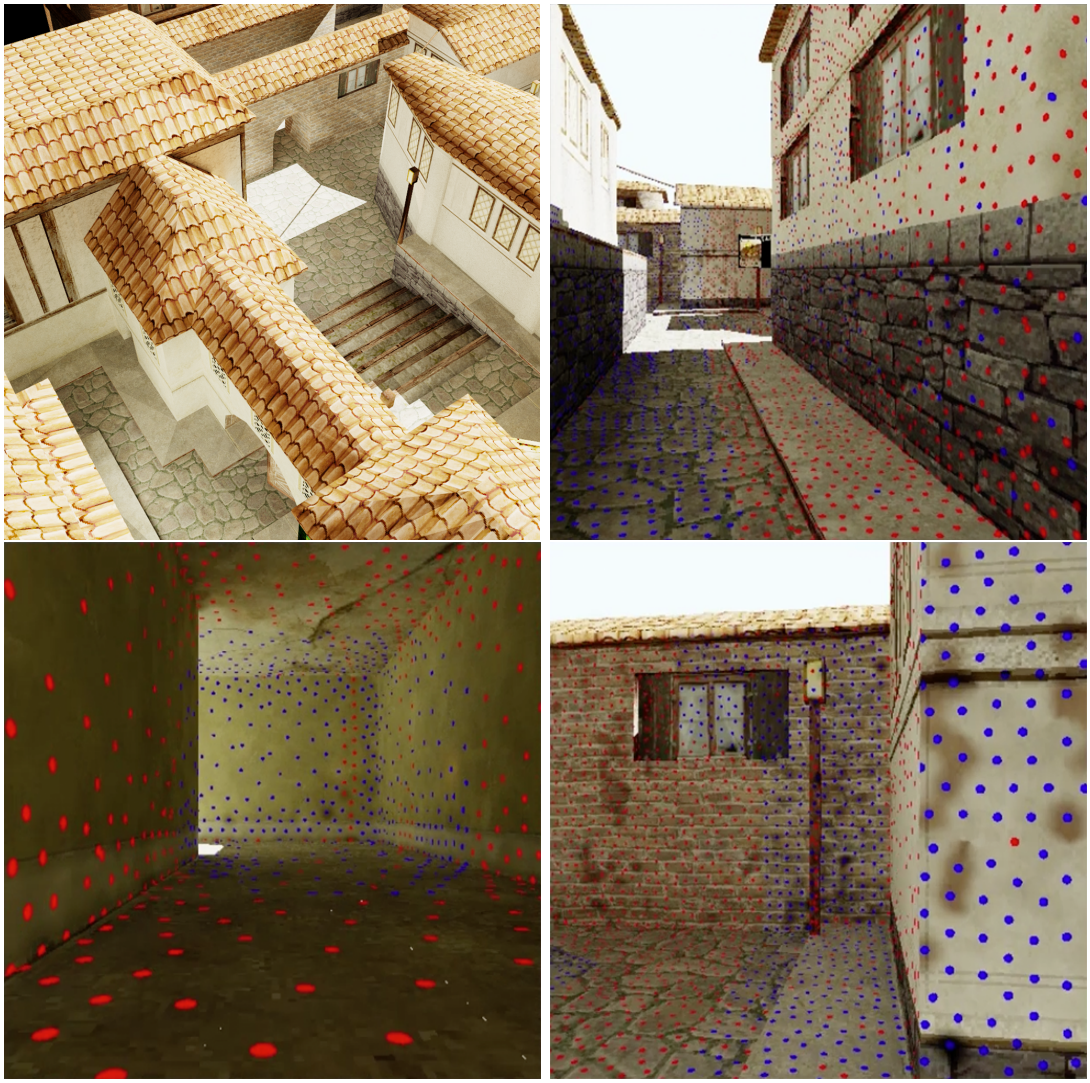


Figure 8.11: Rendered output from Town scene. Irradiance samples are shown in blue and red circles, where blue represents collaboration and red local computation.

agated to all participants making the shared IC consistent at all peers. The IC is considered to have achieved eventual consistency. Speed-up is then measured by having each peer join the quiescent network and perform its respective walkthrough. The peer should require no further changes to its IC once it has been made consistent with that of the other machines on the network. Since connectivity and network management is done in the background, the peer will have generated some irradiance samples before the first exchange is carried out. Table 8.1 shows the time taken for each peer to perform a walkthrough, both as a single machine, without any collaboration, and as a machine joining a quiescent net-

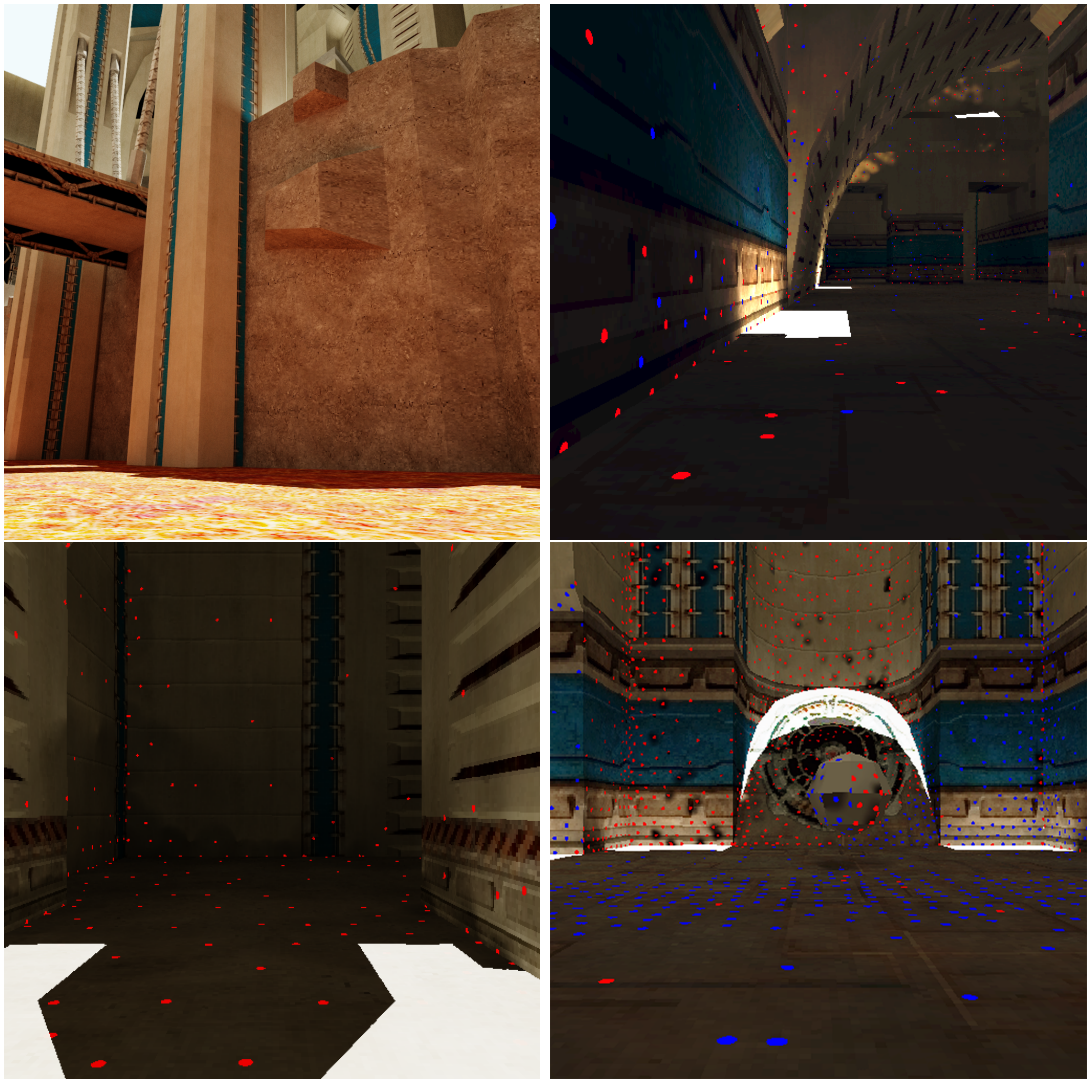


Figure 8.12: Rendered output from Sanctum scene. Irradiance samples are shown in blue and red circles, where blue represents collaboration and red local computation.

work, reflecting the best-case performance where few, if any, irradiance samples are computed by the peer. It should be noted that the path the peer is on would not necessarily have been computed unless similar paths were followed by the other peers.

Peers joining a quiescent network with a saturated IC demonstrate speed-ups between 1.5× and 6× with respect to those generating the IC samples themselves, for the same path. It is important to stress that the speed-up is not the result of distributing work to other peers in the traditional sense, but rather from re-using results from other peers that they would have computed anyway.

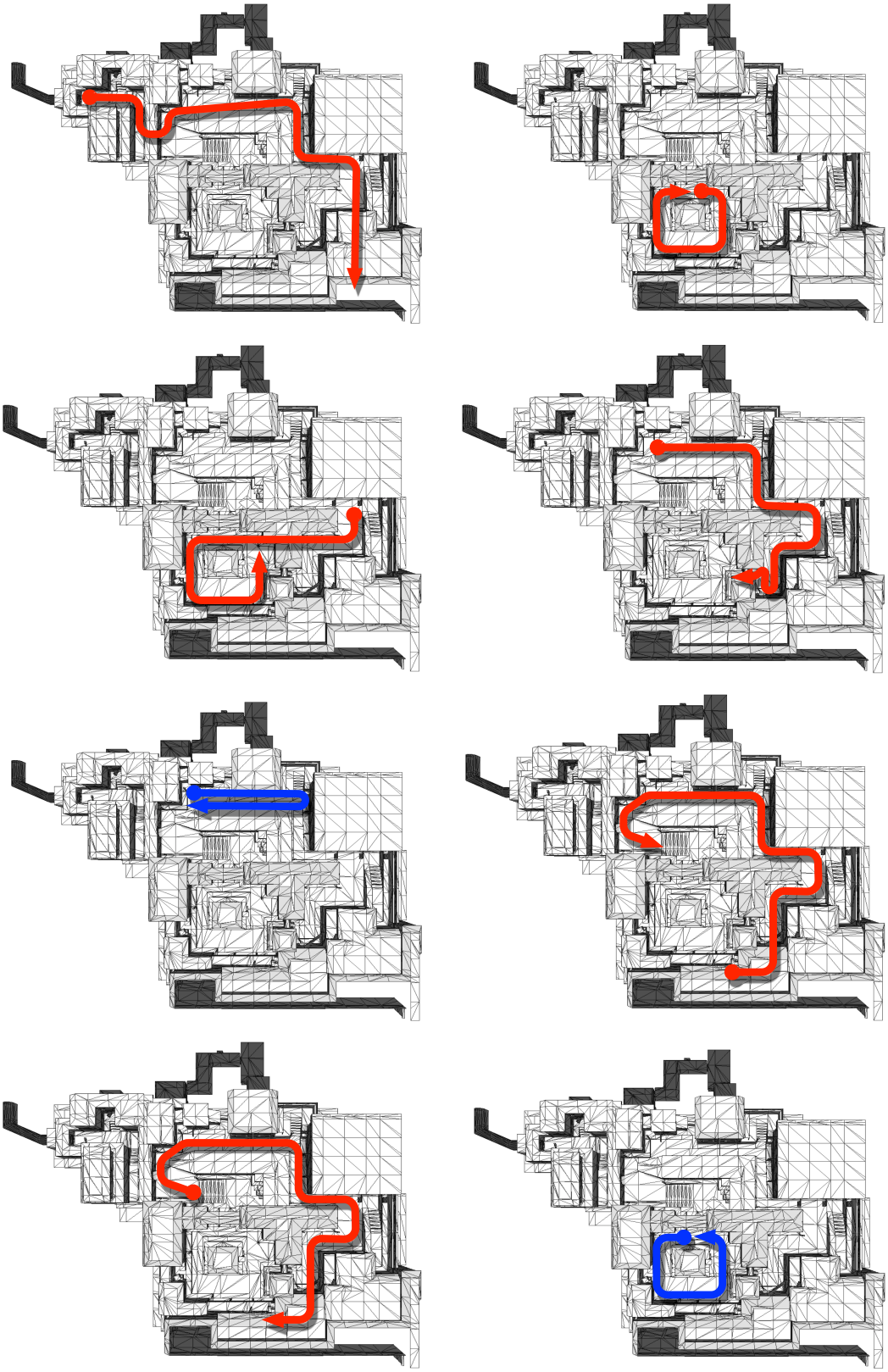


Figure 8.13: Walkthough paths for each peer on Town scene.

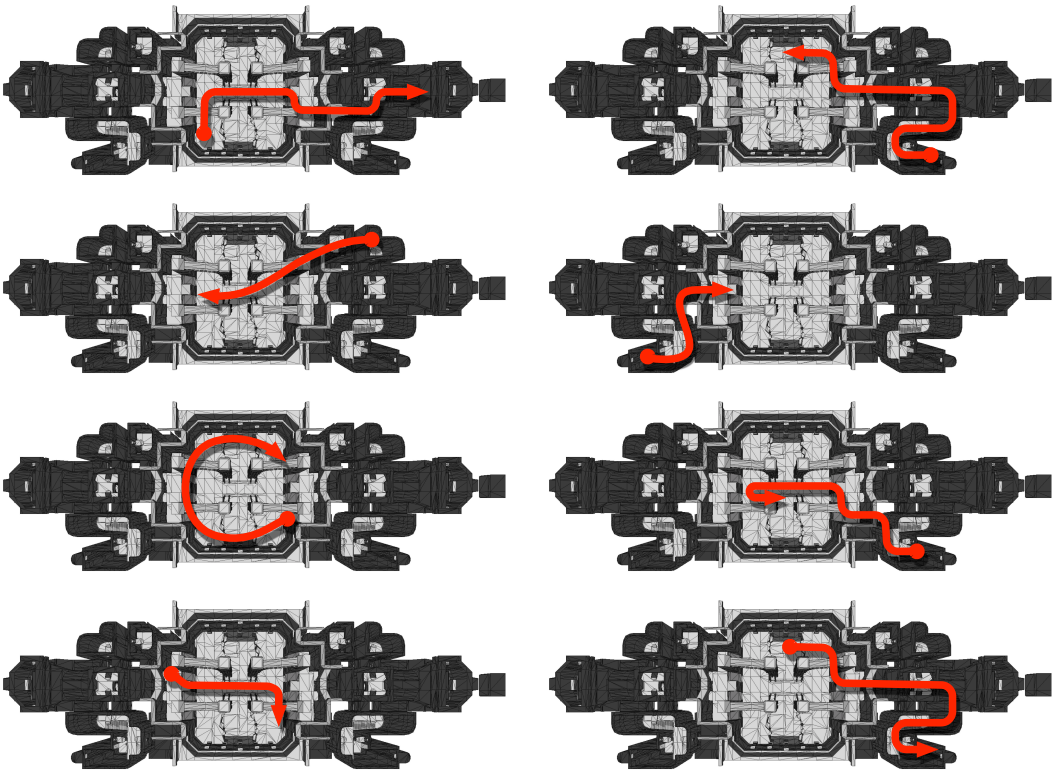


Figure 8.14: Walkthrough paths for each peer on Sanctum scene.

8.4.2 Simultaneous Start

In this test, the effect of the system on rendering times when the network peers boot up simultaneously is observed. The neighbour cache for each peer contains the address of exactly one other machine (§8.2.4), with the initial configuration being that of a daisy-chain. The results are shown in Figure 8.15. In the test, peers find an empty global IC on joining the network. The mean speed-up of the network is around 1.17× for both scenes; when the peers are considered individually, it can be observed that not all of them benefit from this speed-up. Newly generated samples may take time to propagate across the network, and in the meantime peers which could have benefitted from these samples would have computed their own. Also, a peer may only take advantage of topical samples; even if it were to receive a substantial number of irradiance samples from another peer early on in its walkthrough, these samples could only be used if the paths of the two peers intersect each other at some point. Since the best-case rendering times for each peer on both scenes have been determined (see Table 8.1), the results can be contextualised in terms of these values. In Figure

Scene	Peer	Single	Quiescent	Speed-up (×)
Town	1	692	218	3.17
	2	539	226	2.39
	3	402	215	1.87
	4	566	347	1.63
	5	568	313	1.81
	6	344	250	1.38
	7	816	346	2.36
	8	536	284	1.89
Sanctum	1	566	119	4.76
	2	1153	465	2.48
	3	502	105	4.78
	4	513	151	3.40
	5	476	109	4.40
	6	750	123	6.09
	7	866	190	4.56
	8	689	149	4.62

Table 8.1: Worst and best-case rendering times in seconds for tested scenes.

8.15 (top), the light grey bars show the rendering times for the worst case, while the dark bars show the best case. These bounds have been used to compute the speed-up (mid) and efficiency (bottom) for each individual peer. The efficiency is determined by the bound that exists on the speed-up that can be possibly achieved. For instance, if a peer renders a path that is totally disjoint from the path of any other peer on the network, then, meaningful collaboration cannot be expected to take place. Furthermore, no speed-up would be expected, but this would not impinge on the collaboration efficiency for the respective peer. On the other hand, if the same peer has the potential for speeding up its computation hundred-fold but only achieves twenty-fold, then, notwithstanding the results, the collaboration efficiency is relatively low in the particular case. Efficiency is computed by normalising the speed-up by the maximum achievable speed-up. The mean network efficiency, shown as the red horizontal line in the bottom charts, is 59% and 23% for the Town and Sanctum scenes respectively, and this disparity is directly related to the higher maximum speed-up in Sanctum.

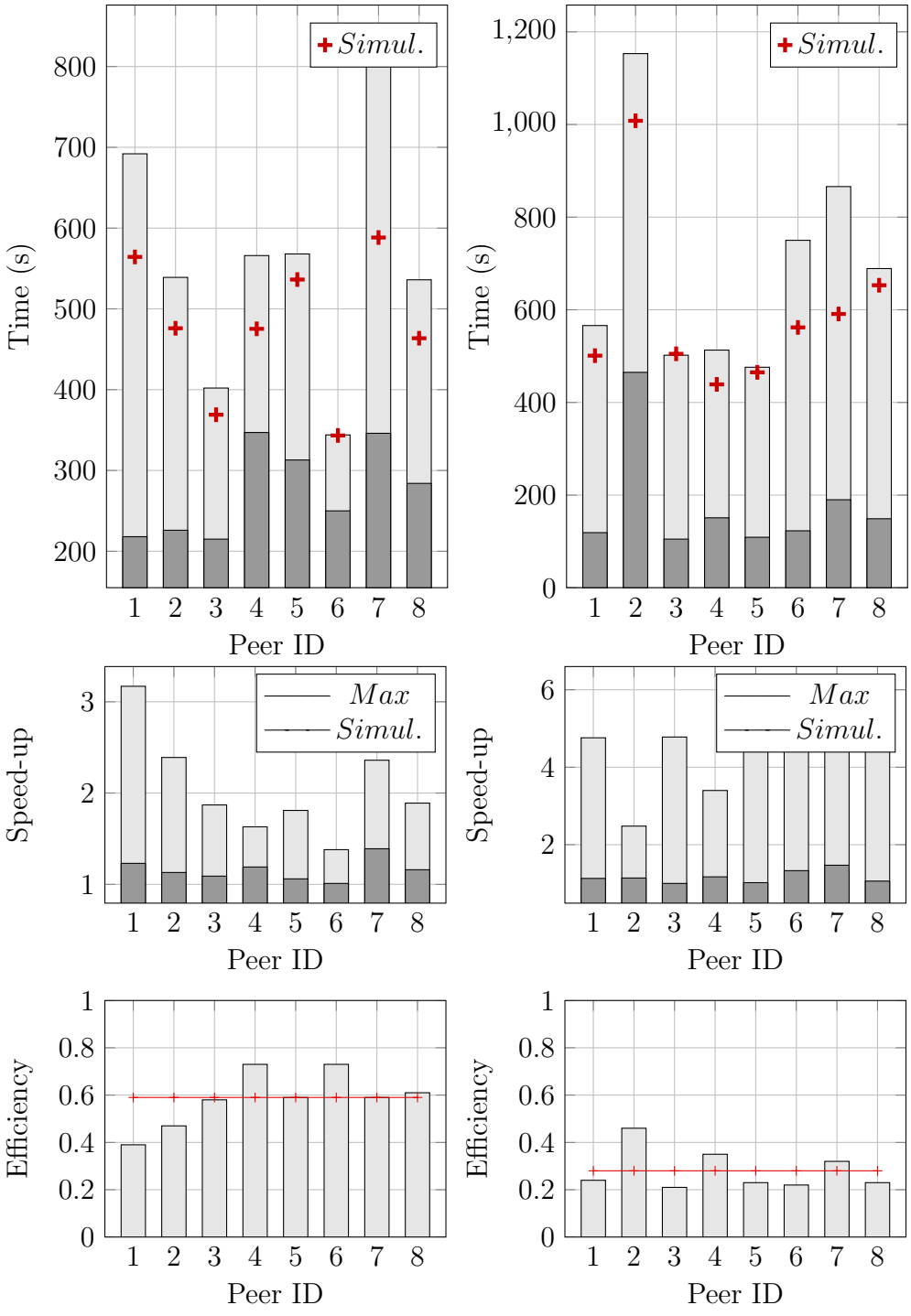


Figure 8.15: Rendering times (top), speed-up (mid) and collaboration efficiency (bottom) for simultaneous start-up on Town (left) and Sanctum (right) scenes.

8.4.3 Staggered Start

The staggered-start tests initially boot the system using a single peer. Additional peers join the network at intervals of $60s$, for the first test (i), and $120s$ for the second (ii), simulating a number of individuals joining a collective group at different times in P2P networks. The results for these tests are shown in figures 8.16 and 8.17 respectively. In the first test, the mean network speed-up was higher than the worst case (by approximately 1.2× both scenes), albeit it differed only marginally from the simultaneous start-up results. In the case of the Town scene, efficiency was unchanged at 59%, but in Sanctum there was an improvement, from 23% to 33%. In the second test, the mean network speed-up was higher, at 1.35× and 2× for Town and Sanctum respectively, and efficiency also, at 36% and 68%.

8.5 Discussion

The results show that rendering times for the IC can be improved using collaborative rendering over P2P networks. The improvement is more marked as less concurrent work is carried out by the peers in the network. Increasing the inter-arrival time between peers joining the network correlated to improvement in the network gains in terms of speed-up. The per-frame rendering performance for each peer fluctuated highly, from 10 frames per second to 5 seconds per frame, depending on the capabilities of the device and the complexity of the view being rendered, as well as the current state of the IC. Although the system was conceived as a means for speeding up interactive rendering, the results clearly show that it falls short of this goal. A problem that stemmed from the use of the IC as a shared data structure, rather than the P2P system per se, is the oversaturation of the octree. Since irradiance samples are generated on-demand, a peer is guaranteed to generate no more samples than it needs, and thus, the irradiance interpolation phase for that frame and subsequent ones will not process unduly redundant data. When irradiance samples generated by concurrent observable events for similar views are merged, it was found that the search performance degrades despite the culling of records in densely populated areas (§8.3.5). A solution to this problem could be the use of adaptive poisson disk sampling, where the radius of rejection adapts to the surface under the point. The current radius is set to the minimum required in order to capture clustered points about edges

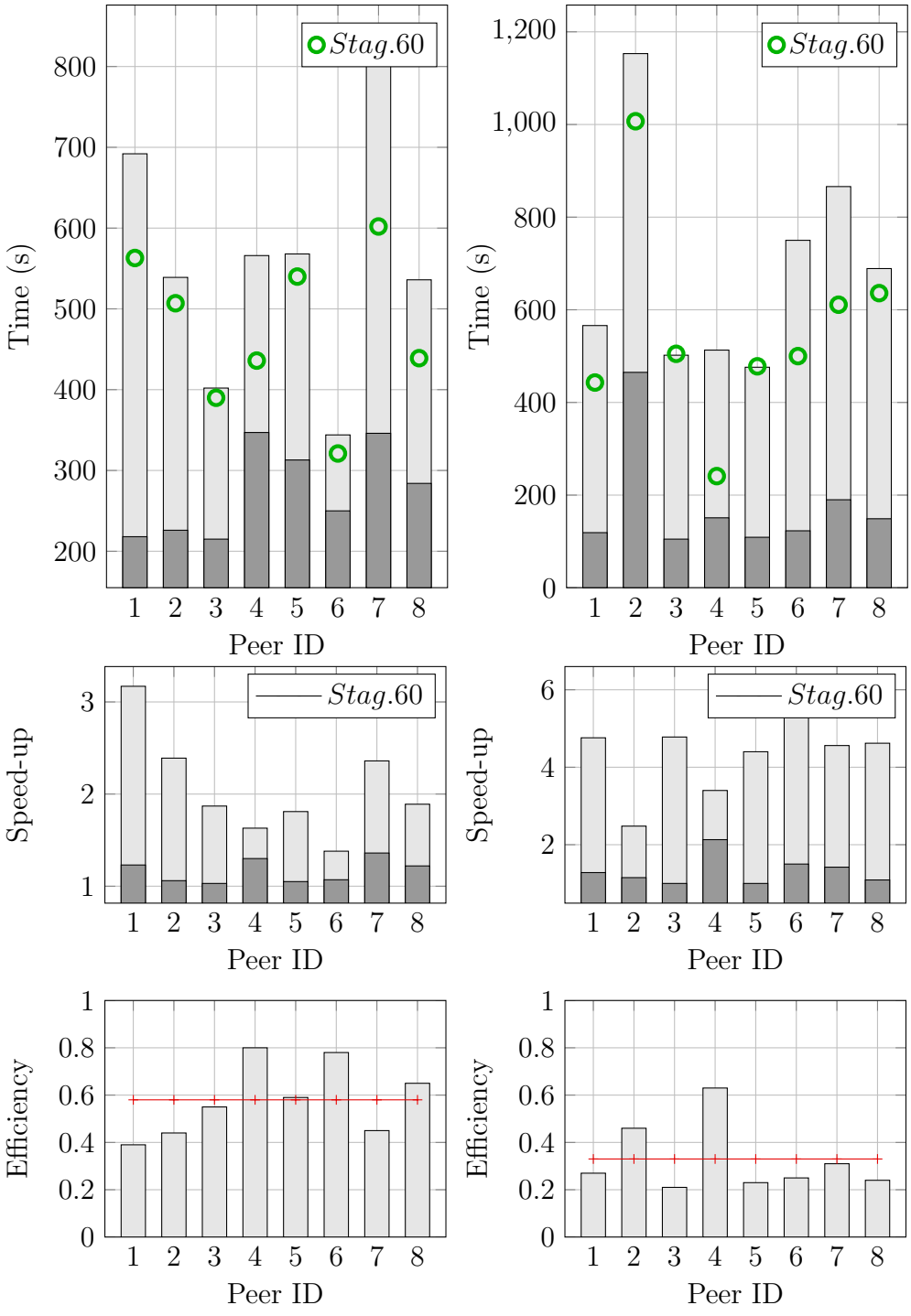


Figure 8.16: Rendering times (top), speed-up (mid) and collaboration efficiency (bottom) for 60s staggered start-up on Town (left) and Sanctum (right) scenes.

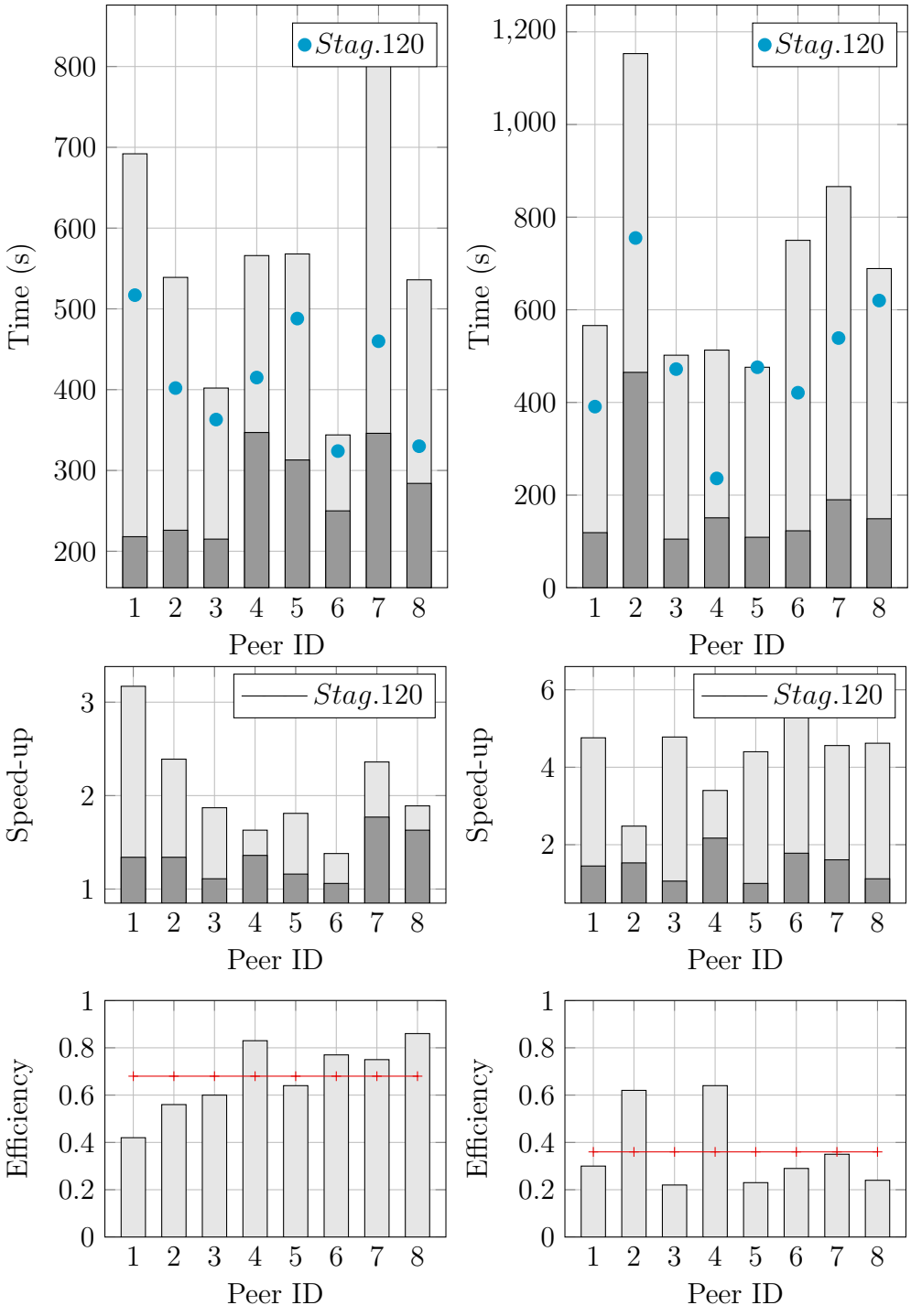


Figure 8.17: Rendering times (top), speed-up (mid) and collaboration efficiency (bottom) for 120s staggered start-up on Town (left) and Sanctum (right) scenes.

and discontinuities; an adaptive radius could contract and expand as required. This leads to the problem of sampling the surface to determine occlusion, which could be an expensive process. This could be solved by sampling the irradiance records already in the cache to determine the local surface curvature. In the case no records are available, then the insert will not result in redundant records in the first place. Since the local IC is a subset of the shared global IC, a peer may hold records generated by other peers that it may never need. This presents a detrimental effect on the performance of the octree traversal during search. Spina *et al.* (2012) accelerate nearest neighbour searches in kd-trees using a two-stage acceleration structure, a regular grid for partitioning the scene combined with a kd-tree at each voxel of the grid.

Another solution, that could be used to simultaneously tackle both issues, is that of using an additional staging octree to hold records received over the network. The original (online) octree is uniquely traversed to interpolate irradiance. When the need for a new sample arises, the staging octree is queried and the new record interpolated from the available ones. If no interpolation is possible, then a new irradiance sample must be computed using ray tracing. Hypothetically, the advantage of this approach is that records added to the online octree on demand. It follows that unvisited areas will not penalise the search, and neither will an oversaturated octree.

The event grouping parameter k (see §8.3.4), was set to 4, which is the size of the local peer cache. The value has been set arbitrarily and requires further study and evaluation in order to ascertain what a good value for the general case would be.

8.6 Summary

This chapter presented PePeR, a novel algorithm for high-fidelity collaborative rendering over P2P networks. PePeR is the natural progression to the work discussed in the previous chapters, taken to be fully decentralised. The reference implementation and respective case study demonstrate that it is possible to take advantage of collaboration in P2P systems to speed up high-fidelity rendering. The novelty of this work lies in laying the foundation for P2P systems for high-fidelity rendering that provide an alternative to the status quo of centralised systems (see Table 9.3).

CHAPTER 9

Conclusions and Future Work

High-fidelity rendering has become an increasingly important field in computer graphics, a field that entertains a diverse cross-section of disciplines and applications, from simulations and architecture, to movies and video games. Widening the availability of high-fidelity rendering by providing access to it from devices that hitherto could not, can only benefit the field and the applications that employ it. Parallel and distributed computing have traditionally bolstered such rendering, and even today, the use of GPUs is invaluable in accelerating image synthesis. Nevertheless, given the largely embarrassingly parallel nature of the computations, the focus has mostly been on ever improving traditional algorithms, while alternative algorithms have received little attention.

This chapter concludes the work presented in this thesis. Unconventional distributed computing paradigms in the field of computer graphics are applied to image synthesis with the aim of furthering the quality of the rendering by accelerating the computation of global illumination. The four representative methods provide interactive high-fidelity rendering to a wide range of computing devices of varied computational power. Chapter 5 presented a system that provides multiple clients with parallel resources for rendering a single task, and adapts in real-time to the number of concurrent requests. Chapter 6 introduced a distributed algorithm for the remote asynchronous computation of the indirect diffuse component, which is merged with locally-computed direct lighting for a full global illumination solution. Chapter 7 detailed a method for precomputing indirect lighting information for dynamically-generated multi-user environments by using the aggregated resources of the clients themselves. Chapter 8 presented a novel peer-to-peer system for improving the rendering performance in multi-user environments through the sharing of computation results, propagated via a

mechanism based on epidemiology.

9.1 Contributions

The overarching contribution of this thesis is demonstrating that true democratisation of high-fidelity rendering is achievable. Distributed computing algorithms that have been largely ignored in the field of computer graphics can be leveraged into improving computation times and bringing high-fidelity rendering to a wide spectrum of computing devices. Table 9.3 provides a feature and performance comparison of the presented methods to related work previously shown in Table 4.5, where it can be seen that the introduction of RaaS, RAIL, PPIL and PePeR into the field of interactive high-fidelity rendering fills existing gaps in the state of the art (Interactive, Scalable, Elastic, M:N; Decentralised, Cooperative) and improves the status quo (High-interactivity, Scalable, Amortised, Low-bandwidth). These methods have shown that adapting techniques from distributed systems can be very successful as long as careful consideration is given to how the methods are applied in order to preserve their core characteristics.

9.1.1 Rendering as a Service (RaaS)

Scalability is intrinsically tied to work in parallel and distributed rendering and all previous work in the field addresses this to some extent. More recent work, however, finds less need for horizontal scaling due to a wider use of GPUs (*OnLive*, 2014; *PlayStation Now*, 2014; Crassin *et al.*, 2013). Particularly, *OnLive* (2014) and *PlayStation Now* (2014) do not try to advance the quality of the rendering beyond what is achievable on a single machine, and the de facto exclusion of horizontal scaling is a design decision rather than a lacuna. Large scale systems such as Patoli *et al.* (2009b), Ramos *et al.* (2009) and Aggarwal *et al.* (2012) scale reasonably well; Patoli *et al.* (2009b) and Ramos *et al.* (2009) do not share the goals of this thesis but target non-interactive offline rendering. Aggarwal *et al.* (2012) support interactive rendering but only for a single client; their system lacks the ability to quick start computations on an idle machine, with jobs taking a few minutes to ramp-up (Aggarwal, 2010). Rapid elasticity, a *sine qua non* of cloud systems, is not addressed by previous work. Introduced in Chapter 5, RaaS provides scalable interactive high-fidelity rendering as a service and rapid elasticity is one of its distinguishing features. The system

modularises stages in a high-fidelity rendering pipeline to provide a flexible and scalable solution that can provide computational resources to a client on demand. RaaS can adapt to its workload by dynamically redistributing the available resources across the connected clients, demonstrating elastic reprovisioning. The response time for resource reprovisioning was shown to be approximately one second. A fine-grained lazy-loading pattern is used in scene initialisation with the aim of reducing response times during reassignment of resources, which further bolsters the elasticity of the system. In order to further improve response times for select algorithms, progressive rendering is applied; a temporal filtering technique is introduced to reduce the disparity between frames at different levels of solution convergence. Results were demonstrated for a number of scenes and high-fidelity rendering algorithms at two different output resolutions, showing reasonable speed-up in all cases. Parallel efficiency for 64 processors was approximately 75%. The Šibenik Cathedral scene was rendered at 30 Hz using Whitted-style ray tracing, and at 25 Hz using IGI-X for a full global illumination solution. The overhead for rendering within the framework was found to be under 2 ms for 128 processors. In terms of actual penalties, this could be as high as 7% of frame time for output at 30 Hz.

The results show that although RaaS can provide interactive high-fidelity visualisation, it cannot sustain high interactivity, and at higher output resolutions it is limited to low interactivity. Resources can be reprovisioned within one second of a request taking place, and although sufficient for the general case, sub-second bursts in workload cannot be handled by the current system. The server-client streaming paradigm takes an approach similar to *PlayStation Now* (2014) and *OnLive* (2014) where output frames are compressed and streamed to the client over the network. This approach is susceptible to network behaviour; in busy networks, the system may suffer from high response times where the displayed output seems to be lagging behind user input. Furthermore, when the network connection between client and server is disrupted, the system becomes inoperable.

List of Contributions

- a specification for scalable and elastic interactive high-fidelity rendering
- a proof of concept for rendering as a service that is flexible, elastic and scalable, for a number of high-fidelity rendering algorithms

9.1.2 Rendering Asynchronous Indirect Lighting (RAIL)

Remote streaming services such as *OnLive* (2014) and *PlayStation Now* (2014) provide a low-bandwidth solution that trades image quality and frame rate. Network stability and traffic can drastically affect input latency, resulting in frame rate drops (Bierton, 2012). Crassin *et al.* (2013) presented a partial streaming method where lighting components are decoupled and independently computed using different methods. Out of the proposed three, as discussed in §4.6, the irradiance maps approach is the most efficient with respect to bandwidth, having similar requirements to *OnLive* (2014) and *PlayStation Now* (2014). However, a laborious precomputation step hamstrings the method, limiting its usability in practice.

RAIL (Chapter 6) leverages the decoupling of different lighting components, an approach developed concurrently to CloudLight. The bandwidth requirements of RAIL are in the same league as *OnLive* (2014) and *PlayStation Now* (2014), and in contrast to CloudLight, it is not limited to a maximum frame rate of 30 Hz. A precomputation stage is required by RAIL, but the process is independent of the complexity of the underlying geometry and totally automated, requiring no user intervention. Similarly to Crassin *et al.* (2013), RAIL supports amortisation of computation over the number of connected clients.

In contrast to *OnLive* (2014) and *PlayStation Now* (2014), RAIL uses asynchronous computation to decouple indirect lighting - computed on a powerful server - from the rest of the rendering, eliminating the problem of input latency at the cost of a slightly higher computational bar at the client side. The use of an object-space representation in the computation of indirect lighting allows the system to scale easily over the connected clients. Furthermore, it facilitates the streaming of a lightweight representation of data when information is exchanged with the clients. This representation is independent of display resolution and thus, bandwidth requirements and general communication costs do not increase with improvements in image definition. Results have shown that the bandwidth requirements of RAIL do not exceed those of streaming cloud gaming services such as *OnLive* (2014) or *PlayStation Now* (2014), while at the same time allowing higher output resolutions without additional communication costs (see §6.4.2). Such systems require a very good network connection to stream output at 60 Hz; network fluctuations can introduce variable input response times that may prove jarring to a user. In RAIL the perceptually salient direct lighting

component is computed locally, thus removing any possible lag due to poor network availability. In addition, the client reconstruction process has been shown to be both lightweight and scalable, and well-suited for any device, from tablet to desktop machine. RAIL can operate at reduced visual quality when network services are disrupted.

RAIL introduces a novel multi-bounce GI algorithm that is suitable for applications requiring low input latency and high interactivity. The method only simulates diffuse indirect lighting, making it less accurate than the traditional methods employed in RaaS and in other works. The higher order ambient function proposed as part of this algorithm is also a coarse approximation of indirect lighting for animated objects in the scene that is more ad hoc than physically-correct, and impinges on the correctness of the overall solution.

List of Contributions

- a novel fast algorithm for multi-bounce global illumination based on instant radiosity methods
- a scalable asynchronous distributed rendering algorithm with low bandwidth requirements that is resolution-independent and robust to network service fluctuations
- a novel higher order ambient function for approximating diffuse indirect lighting
- the application of RAIL to RaaS for highly interactive, high-fidelity rendering in HD (and higher), over a wide spectrum of devices

9.1.3 Precomputed Per-vertex Indirect Lighting (PPIL)

Previous decentralised approaches to high-fidelity rendering are few and only concern offline rendering. Ramos *et al.* (2009) propose a partially decentralised architecture where asset synchronisation between peers was carried out using P2P file sharing, while the actual rendering utilised a master-worker approach. The approach taken by PPIL also partially decentralised, wherein a group of client devices elect a master to coordinate the precomputation of indirect lighting, for use in real-time rendering. The main computation gains come from amortisation.

PPIL (Chapter 7) scales the method used in RAIL, and applies it to the precomputation of online per-vertex indirect lighting for dynamically-generated scenes. The focus of PPIL is the augmentation of rendering for devices that benefit from simple hardware acceleration; the method is meant to be non-intrusive and integrable in standard rendering systems. PPIL is applicable to a broad range of devices, but is especially targeted towards those lacking computational power, such as entry-level tablets and smartphones. The precomputation process takes advantage of the aggregated computational power of the network formed by the connected devices. An elected master device partitions scene geometry and distributes these partitions in terms of work to the other participants. The master-worker paradigm is employed. Lighting information is computed and stored at geometry vertices. At runtime this information can interpolated and merged with direct lighting operations to obtain a GI solution. The results demonstrate that through this method image realism is drastically enhanced when compared to a constant ambient lighting function. This is achieved at a negligible cost in runtime performance. Although PPIL is a solution for dynamically-generated environments, the indirect lighting information cannot change after it is precomputed, making it unsuitable for dynamic scenes. The quality of the indirect lighting solution is dependent on the level of detail of scene geometry, since indirect lighting is sampled at the vertices. This is a common limitation of many rendering methods; low-detail geometry may lead to under-sampling of lighting and in turn aliasing, causing the solution to miss important features of light distribution in the environment.

List of Contributions

- a scaled version of the global illumination algorithm in RAIL, adapted for per-vertex indirect lighting, that is suitable for devices with basic hardware rendering capabilities
- a distributed algorithm for precomputing indirect lighting information that is suitable for dynamically-generated environments

9.1.4 Peer-to-peer Rendering (PePeR)

In Chapter 8, peer-to-peer collaborative rendering was introduced, which reduces redundancy of computation in unstructured P2P networks. PePeR is especially

relevant in multi-user virtual environments where collaborators interact over similar areas in a shared environment. An event system is used to sequence access to shared data structures which capture visual features of these environments, such as lighting. Events are propagated across the unstructured network using algorithms inspired from epidemiology. A peer randomly chooses another from a local directory and performs an anti-entropy reconciliation step to harmonise known events. Peer directories are updated through a merging and sorting process. PePeR has the advantage of being totally decentralised and any computation gains experienced by the peers are effected through amortisation rather than farming of computation: speed-up comes at little to no cost, since peers would have performed the respective computations anyway. The results show the application of PePeR to the irradiance cache algorithm, evaluated on two large scenes from competitive gaming. Eight peers were used in the evaluation, with individual per-peer speed-up of up to 6× and combined network speed-up ranging from 1.2× to 2×.

PePeR provides a means of abstracting access to shared data structures and its performance as a rendering method is highly dependent on the actual visualisation algorithm used. While the IC stores diffuse indirect lighting information that can be reused by other peers, the actual computations generate spikes in frame workloads that force the rendering into the low interactivity band. Moreover, the dependence of update latency on the event propagation system means that frequent updates may oversaturate the network needlessly or force peers into performing duplicate computations for events that haven't propagated to them yet.

List of Contributions

- the application of peer-to-peer to high-fidelity rendering
- the introduction of an event-based system for encapsulating ordered access to a shared data structure
- the application of an epidemiological method for event propagation within an unstructured network
- a novel collaboration algorithm for high-fidelity rendering

- the application of PePeR to high-fidelity rendering using the irradiance cache

9.2 Significance

Image synthesis and high-fidelity rendering in particular are computationally expensive processes that require dedicated resources such as powerful GPUs to visualise even moderately complex environments. Despite its large area of applicability, there are an ever-growing number of devices such as tablets and smartphones that are precluded from high-fidelity visualisation (see Chapter 1). There has been little divergence from the traditional application of distributed computing approaches to rendering, most of which were targeted for dedicated systems and render farms. The recent push towards service-based approaches such as cloud-based render farms and streaming online services (*renderRocket*, 2014; *PlayStation Now*, 2014; Crassin *et al.*, 2013) highlights the significance of novel forms of distributed computing in computer graphics and rendering in particular. Notwithstanding the novelty of such approaches, the overwhelming majority of research in the area gravitates towards centralisation of resources, with interactivity and rendering quality being heavily dependent on the physical link between client and server. Even with most devices possessing Internet connectivity in some form or other, research has veered away from the application of paradigms such as peer-to-peer. This work was undertaken to find out whether unconventional distributed rendering techniques can provide a valid alternative to, or possibly complement, more conventional techniques in delivering high-fidelity rendering to devices with different characteristics and capabilities.

9.2.1 Findings

High-fidelity rendering can be offered to a variety of devices at low to medium interactive rates, as shown in Chapter 5 (see also §5.6, §9.1.1). All things being equal, accuracy of the rendering solution and output resolution are primary factors in determining server-side load and resource usage, as suggested by the results in §5.6.1. Sacrificing accuracy and visual fidelity can help achieve higher frame rates although not necessarily low input latency, which is dependent on network characteristics and communication requirements. When high interactivity is more important than solution quality, partially offloading the rendering com-

putations to the client can help reduce input latency. In Chapter 6, it was shown that high interactivity and low input latency can be achieved through the application of a less accurate asynchronous global illumination algorithm, distributed between client and server (see §6.4). Moreover, during data exchange between server and client, storing lighting information in object space rather than screen space can help reduce bandwidth requirements, in some cases by an order of magnitude when compared to previous work, even for HD or higher resolutions (see §6.4.2). For multi-user virtual environments, object space representations help amortise the costs of indirect lighting computation across multiple clients. In multi-user environments, cooperation can be leveraged for high-fidelity visualisation in the absence of a central powerful server. Through the use of an online precomputation step that is shared by all collaborators, static indirect lighting can be used to augment visual fidelity at a marginal increase in runtime costs, as shown in Chapter 7 (see §9.1.3, §7.4). In Chapter 8 it was demonstrated that collaborative rendering can also be employed without invoking a precomputation step, in a fully decentralised network (see §8.4). The abstraction of global state over a P2P network has been shown to give multi-user environments a framework for sharing computation and mitigating redundancy. Amortised speed-up has been recorded, with frame rates in the low interactivity range. The latency of updates through sharing of computation is dependent on the frequency of anti-entropy exchanges between peers and the size of the network. In a large network, information will take longer to propagate and thus, update latency is expected to be higher.

From these findings it is clear that in addition to the trade-off between interactivity and visual fidelity in rendering, distributed algorithms extend this interplay to latency arising from network communication. Table 9.3 gives a breakdown of the methods presented in terms of their latency, interactivity and visual fidelity. An ideal distributed high-fidelity rendering algorithm would score high in both interactivity and fidelity, while keeping latency as low as possible. The table also suggests that the methods presented in this work are not equivalent, possessing distinct characteristics, and may be targeted towards specific areas of application, as discussed below.

Method	Chapter	Latency	Interactivity	Fidelity
Ideal	-	Low	High	High
RaaS	5	Mid	Mid	High
RAIL	6	Low	High	Mid
PPIL	7	Low	Mid	Low
PePeR	8	High	Low	High

Table 9.1: Performance in terms of latency, interactivity and visual fidelity.

9.2.2 Impact

The idea of rendering as a service could prove beneficial to architects, designers and engineers who would want ubiquitous high-quality visualisation of their work that can be manipulated in real-time. RaaS, as a paradigm, can benefit small start-ups that cannot afford the upfront costs of setting up dedicated clusters for such rendering. RAIL shares many properties with RaaS, with the added benefit of supporting high-interactivity at the cost of a less accurate solution. A field that can be heavily impacted by the introduction of RAIL is that of entertainment, where it could be applied to augment realism of video games at a small realisation cost, which is outweighed by the resulting quality. In particular, RAIL could enhance immersion of games running on tablets and previous generation video game consoles which are not computationally equipped to provide an equivalent GI solution. RAIL could also benefit current generation consoles by offloading indirect lighting computation and channelling free processor time into providing better physical simulations of object behaviour or improved artificial intelligence (AI) of non-player characters. PPIL could also be advantageous within the realm of video games, enhancing quality on computationally weak devices. In the serious games community, it could be used to augment the realism of visualisation for training simulators that feature multi-user environments, such as some educational applications that are meant to run on a wide variety of platforms and usually focus on collaborative interactions. PePeR could have a more extensive impact than the other proposed methods since its application lends itself to areas outside of computer graphics. In a more broad and general sense, PePeR can be applied to data sharing and collaboration within volatile environments where fault-tolerance is important and churn is a reality. Within computer graphics, it can be used to speed up both interactive and offline high-fidelity rendering.

Although this thesis focused on high-fidelity rendering, the work presented

Method	Reference
Keates & Hubbold (1995)	[1]
Muuss (1995)	[2]
Parker *et al.* (1998)	[3]
Parker *et al.* (1999)	[4]
Wald, Kollig, Benthin, Keller & Slusallek (2002)	[5]
Rangel-Kuoppa *et al.* (2003)	[6]
Patoli *et al.* (2009a)	[7]
Ramos *et al.* (2009)	[8]
Gonzalez-Morcillo *et al.* (2010)	[9]
Pajak *et al.* (2011)	[10]
Aggarwal *et al.* (2012)	[11]
Crassin *et al.* (2013), Irradiance maps	[12*a*]
Crassin *et al.* (2013), Voxel cone tracing	[12*b*]
Crassin *et al.* (2013), Photon tracing	[12*c*]
OnLive (2014)	[13]
PlayStation Now (2014)	[14]
RaaS (Chapter 5)	[15]
RAIL (Chapter 6)	[16]
PPIL (Chapter 7)	[17]
PePeR (Chapter 8)	[18]

Table 9.2: Reference to work in parallel and distributed rendering including the methods presented in this work.

can be extended to other areas of computer graphics. Unconventional distributed algorithms may be gainfully employed in other areas such as non-photorealistic rendering, or more broadly, physical simulation and digital imaging, which could benefit from collaboration, speed-up or a service-based approach.

9.3 Limitations and Future Work

This section outlines some limitations of the work presented in this thesis together with possible avenues for future work. Future work in the areas of high-fidelity rendering and distributed computing could address the limitations of the presented algorithms.

Fault-tolerance is an important aspect of service oriented architectures; in this work, RaaS and PPIL, which are the main service-oriented methods presented, do not actively focus on providing fault-tolerance. Both methods are susceptible

Method	Scalable	Elastic	Interactive	M:N	Decentralised	Cooperative	Speed-up	Bandwidth	Precomputation	Resolution
[1]	✓	—	Low	—	—	—	✓	—	—	—
[2]	✓	—	Low	—	—	—	✓	—	—	486p
[3]	✓	—	Low	—	—	—	✓	—	—	512p
[4]	✓	—	Low	—	—	—	✓	—	—	512p
[5]	✓	—	✓	—	—	—	✓	—	—	480p
[6]	✓	—	✓	—	—	—	✓	—	—	—
[7]	✓	—	—	✓	—	—	✓	—	—	—
[8]	✓	—	—	✓	Partial	—	✓	—	—	—
[9]	✓	—	—	✓	Partial	—	✓	—	—	—
[10]	—	—	✓	—	—	—	✓	Scalable	—	—
[11]	✓	—	✓	—	—	—	✓	—	—	—
[12*a*]	Vertical	—	✓	—	—	—	Amortised	Low	✓	1080p
[12*b*]	Vertical	—	✓	—	—	—	Amortised	Medium	—	1080p
[12*c*]	Vertical	—	✓	—	—	—	Amortised	High	—	1080p
[13]	Vertical	—	High	—	—	—	—	Low	—	720p
[14]	Vertical	—	✓	—	—	—	—	Low	—	720p
[13]	✓	✓	✓	✓	—	—	✓	—	—	1024p
[14]	✓	✓	High	✓	—	—	Amortised	Low	✓	1080p
[15]	✓	—	✓	—	Partial	✓	Amortised	—	✓	—
[16]	✓	—	Low	—	✓	✓	Amortised	—	—	—

Table 9.3: Feature and performance comparison of parallel and distributed rendering systems.

to single point of failure problems and future work would look into efficient ways of making these algorithms more robust in this respect.

The object lazy-loading mechanism employed in RaaS does not provide a means to quickly update repository objects while ensuring memory consistency across resources. This highlights a limitation within the current implementation of RaaS where deformable objects have to be handled as special cases and cannot be shared across resources, but have to be updated per worker. Further research is essential in order to extend the lazy-loading mechanism to a fully-fledged distributed shared memory. Future work would also investigate the use of PePeR within RaaS, as a means of sharing data structures in a system where resource churn is common. A further limitation of the lazy-loading mechanism which hasn't been formally evaluated is read contention of the object repository, albeit the use of per-job repositories could mitigate the effect this might have on concurrent jobs. Future work would investigate the extent to which this affects the performance of RaaS and ways to reduces contention.

The widespread adoption of GPUs and their use in stochastic ray tracing suggests that the rendering algorithms implemented in RaaS could benefit from the use of GPUs. Future work would look into achieving high-interactivity by adding vertical scaling support for GPUs. To some extent, this approach has been validated by RAIL already, although in this case, research would be looking at traditional high-fidelity image synthesis algorithms which run entirely server-side.

The server-side shading used in RAIL assumes diffuse surfaces; a more complete global illumination solution should cater for phenomena like caustics and glossy surfaces. Research into extending RAIL to augment the shading strategy in an efficient way that would keep bandwidth requirements to a minimum is a promising avenue for future work.

Further research would also look into the request-response model used to synchronise indirect lighting and investigate support for a push-model, which could reduce round-trip latency and provide faster updates to indirect lighting. The VPL tracing pass and the point set shading could be improved by using rasterisation methods in conjunction with ray tracing (McGuire & Luebke, 2009). Future work would also look into extending the coarse grid that approximates indirect lighting for dynamic objects to store directional information, using approaches similar to Greger *et al.* (1998).

Future work into PePeR would look into the implementation of different high-fidelity algorithms and their applicability to a P2P environment. Similarly to

RaaS, PePeR does not take advantage of GPU computing, which would be an interesting avenue to explore, possibly employing the GI algorithm used in RAIL, which is similar in concept to the IC with the added benefits that irradiance samples are decided a priori, and are thus the same for every peer, eliminating any ambiguity that may arise in the merging process for insertion events. The next logical step in the evolution of the system is high-interactivity rendering; to this end, future work should further investigate the application of rumour spreading to the communication of observable events as soon as they are generated.

9.3.1 Future Work in Rendering and Distributed Systems

High-fidelity rendering is always striving to improve rendering models to capture more of the physical properties of lighting and simulate complex natural phenomena. Although offline and interactive rendering are each driven by predominant paradigms, they are highly interrelated areas where models from offline rendering are eventually distilled to real-time rendering by empirical simplification or improvements in hardware that make them feasible for interactive purposes, or conversely, optimisations employed in real-time rendering are integrated into offline techniques to accelerate rendering times. A rich repository of techniques exist in both camps that could be investigated and used to augment some counterpart in the opposite camp. Still, the more immediate challenge that needs to be overcome is that of high-quality global illumination at highly interactive rates.

In the longer term, more accurate models for computer graphics may need to break from a strictly geometric optics modelling of light transport and focus more on the dual nature of light as a wave and a particle. The nature of quantum mechanics, which photons are subject to, may need to be further investigated.

The nature of high-fidelity rendering as a simulation means that any complex models that are grafted from the fields of physics will greatly benefit from parallel and distributed computation - indeed, it is concurrent computing in its various forms that makes these simulations possible. Whether GPUs, supercomputers or distributed systems, computer graphics will always benefit from parallel computation; the challenge is designing efficient parallel and distributed algorithms to maximise these benefits.

Cloud technologies present a new set of challenges for interactive high-fidelity rendering, where algorithms must take into consideration heterogeneous devices, disparate node locations, quality of network service and other infrastructural

issues, but at the same time, they give rise to distributed paradigms that can be adopted into high-fidelity rendering to enable further productivity to be achieved and potentially open up novel uses of computer graphics.

9.4 Final Remarks

The field of computer graphics has seen staggering advances towards realistic and correct image-synthesis, while distributed computing has witnessed unprecedented growth due to the ubiquity of computing brought by mobile technologies and the Internet. The work in this thesis contributes to the body of knowledge intersecting computer graphics and distributed computing in presenting unconventional distributed methods that couple these fields to bring high-fidelity graphics to a vast spectrum of devices, from mobile phones to tablets to desktop computers. The presented algorithms are unconventional by the standards of computer graphics, but widespread in other areas, especially those of service oriented architectures, distributed databases and resource sharing. RaaS provides high-fidelity interactive rendering as a service. RAIL is an extension that provides highly interactive rendering that is independent of network stability and traffic. PPIL scales RAIL to provide indirect lighting to dynamically-generated environments, while PePeR pioneers the concept of P2P collaborative rendering. This thesis has addressed a number of important research challenges faced in the democratisation of high-fidelity rendering, providing a firm foundation from which future research can build.

Bibliography

Aggarwal, V. (2010). *High-fidelity Rendering on Shared Resources*, PhD thesis, University of Warwick.

Aggarwal, V., Chalmers, A. & Debattista, K. (2008). High-fidelity rendering of animations on the grid: a case study, *Proceedings of the 8th Eurographics conference on Parallel Graphics and Visualization*, Eurographics Association, pp. 41–48.

Aggarwal, V., Debattista, K., Bashford-Rogers, T., Dubla, P. & Chalmers, A. (2012). High-fidelity interactive rendering on desktop grids, *IEEE Computer Graphics and Applications* **32**(3): 24–36.

Aggarwal, V., Debattista, K., Dubla, P., Bashford-Rogers, T. & Chalmers, A. (2009). Time-constrained high-fidelity rendering on local desktop grids, *Proceedings of the 9th Eurographics conference on Parallel Graphics and Visualization*, Eurographics Association, pp. 103–110.

Akenine-Moller, T., Haines, E. & Hoffman, N. (2008). Real-time rendering.

Amanatides, J., Woo, A. *et al.* (1987). A fast voxel traversal algorithm for ray tracing, *Proceedings of EUROGRAPHICS*, Vol. 87, pp. 3–10.

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities, *Proceedings of the April 18-20, 1967, spring joint computer conference*, ACM, pp. 483–485.

Anderson, D. P. (2004). Boinc: A system for public-resource computing and storage, *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, IEEE, pp. 4–10.

Andrews, G. R. (1991). *Concurrent programming: principles and practice*, Benjamin/Cummings Publishing Company.

Appel, A. (1968). Some techniques for shading machine renderings of solids, *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, ACM, pp. 37–45.

Ashikhmin, M. & Shirley, P. (2000). An anisotropic phong brdf model, *Journal of graphics tools* **5**(2): 25–32.

Autodesk 360 (2014).
URL: *http://www.autodesk.com/products/rendering/overview*

Aydin, T. O., Mantiuk, R., Myszkowski, K. & Seidel, H.-P. (2008). Dynamic range independent image quality assessment, *ACM Transactions on Graphics (TOG)*, Vol. 27, ACM, p. 69.

Badouel, D. & Priol, T. (1992). An efficient parallel ray tracing scheme for highly parallel architectures, *Advances in Computer Graphics Hardware V*, Springer, pp. 93–106.

Baharon, M. R., Shi, Q., Llewellyn-Jones, D. & Merabti, M. (2013). Secure rendering process in cloud computing, *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*, IEEE, pp. 82–87.

Bailey, N. T. *et al.* (1975). *The mathematical theory of infectious diseases and its applications*, Charles Griffin & Company Ltd, 5a Crendon Street, High Wycombe, Bucks HP13 6LE.

Banino, C. (2006). Optimizing locationing of multiple masters for master-worker grid applications, *Applied Parallel Computing. State of the Art in Scientific Computing*, Springer, pp. 1041–1050.

Banterle, F., Artusi, A., Debattista, K. & Chalmers, A. (2011). *Advanced high dynamic range imaging: theory and practice*, CRC Press.

Bashford-Rogers, T., Debattista, K. & Chalmers, A. (2013). Importance driven environment map sampling.

Baum, D. R. & Winget, J. M. (1990). Real time radiosity through parallel processing and hardware acceleration, *ACM SIGGRAPH Computer Graphics*, Vol. 24, ACM, pp. 67–75.

Bavoil, L., Sainz, M. & Dimitrov, R. (2008). Image-space horizon-based ambient occlusion, *ACM SIGGRAPH 2008 talks*, ACM, p. 22.

Benthin, C., Wald, I. & Slusallek, P. (2003). A scalable approach to interactive global illumination, *Computer Graphics Forum*, Vol. 22, Wiley Online Library, pp. 621–630.

Bierton, D. (2012). Face-off: Gaikai vs. onlive.
URL: *http://www.eurogamer.net/articles/digitalfoundry-face-off-gaikai-vs-onlive*

Bikker, J. & Reijerse, R. (2009). A precalculated point set for caching shading information, *Eurographics 2009-Short Papers*, The Eurographics Association, pp. 65–68.

Blinn, J. F. & Newell, M. E. (1976). Texture and reflection in computer generated images, *Communications of the ACM* **19**(10): 542–547.

Blythe, D., Grantham, B., McReynolds, T. & Nelson, S. R. (1999). Advanced graphics programming techniques using opengl.

Brouillat, J., Gautron, P. & Bouatouch, K. (2008). Photon-driven irradiance cache, *Computer Graphics Forum*, Vol. 27, Wiley Online Library, pp. 1971–1978.

Buck, B. & Keleher, P. (1998). Locality and performance of page-and object-based dsms, *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International... and Symposium on Parallel and Distributed Processing 1998*, IEEE, pp. 687–693.

Caflisch, R. E. (1998). Monte carlo and quasi-monte carlo methods, *Acta numerica* **7**: 1–49.

Catmull, E. (1974). A subdivision algorithm for computer display of curved surfaces., *Technical report*, DTIC Document.

Chai, C. (2002). Consistency issues in distributed shared memory systems.

Chalmers, A. & Debattista, K. (2009). Level of realism for serious games, *Games and Virtual Worlds for Serious Applications, 2009. VS-GAMES'09. Conference in*, IEEE, pp. 225–232.

Chalmers, A., Reinhard, E. & Davis, T. (2002). *Practical parallel rendering*, CRC Press.

Cohen, M. F., Chen, S. E., Wallace, J. R. & Greenberg, D. P. (1988). A progressive refinement approach to fast radiosity image generation, *ACM SIGGRAPH Computer Graphics*, Vol. 22, ACM, pp. 75–84.

Cook, R. L., Porter, T. & Carpenter, L. (1984). Distributed ray tracing, *ACM SIGGRAPH Computer Graphics*, Vol. 18, ACM, pp. 137–145.

Cook, R. L. & Torrance, K. E. (1982). A reflectance model for computer graphics, *ACM Transactions on Graphics (TOG)* **1**(1): 7–24.

Cox, M. & Ellsworth, D. (1997). Application-controlled demand paging for out-of-core visualization, *Proceedings of the 8th conference on Visualization'97*, IEEE Computer Society Press, pp. 235–ff.

Crassin, C., Luebke, D., Mara, M., McGuire, M., Oster, B., Shirley, P., Sloan, P.-P. & Wyman, C. (2013). Cloudlight: A system for amortizing indirect lighting in real-time rendering.

Crassin, C., Neyret, F., Sainz, M., Green, S. & Eisemann, E. (2011). Interactive indirect illumination using voxel cone tracing, *Computer Graphics Forum*, Vol. 30, Wiley Online Library, pp. 1921–1930.

Crockett, T. W. (1997). An introduction to parallel rendering, *Parallel Computing* **23**(7): 819–843.

Dachsbacher, C., Křivánek, J., Hašan, M., Arbree, A., Walter, B. & Novák, J. (2014). Scalable realistic rendering with many-light methods, *Computer Graphics Forum*, Vol. 33, Wiley Online Library, pp. 88–104.

Dachsbacher, C. & Stamminger, M. (2005). Reflective shadow maps, *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM, pp. 203–231.

Dachsbacher, C., Stamminger, M., Drettakis, G. & Durand, F. (2007). Implicit visibility and antiradiance for interactive global illumination, *ACM Transactions on Graphics (TOG)*, Vol. 26, ACM, p. 61.

Dammertz, H., Keller, A. & Lensch, H. P. (2010). Progressive point-light-based global illumination, *Computer Graphics Forum*, Vol. 29, Wiley Online Library, pp. 2504–2515.

Debattista, K., Dubla, P., Banterle, F., Santos, L. P. & Chalmers, A. (2009). Instant caching for interactive global illumination, *Computer Graphics Forum*, Vol. 28, Wiley Online Library, pp. 2216–2228.

Debattista, K., Dubla, P., Peixoto dos Santos, L. P. & Chalmers, A. (2011). Wait-free shared-memory irradiance caching, *Computer Graphics and Applications, IEEE* **31**(5): 66–78.

Debattista, K., Santos, L. P. & Chalmers, A. (2006). Accelerating the irradiance cache through parallel component-based rendering, *Proceedings of the 6th Eurographics conference on Parallel Graphics and Visualization*, Eurographics Association, pp. 27–35.

Deering, M., Winner, S., Schediwy, B., Duffy, C. & Hunt, N. (1988). The triangle processor and normal vector shader: a vlsi system for high performance graphics, *ACM SIGGRAPH Computer Graphics*, Vol. 22, ACM, pp. 21–30.

Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D. & Terry, D. (1987). Epidemic algorithms for replicated database maintenance, *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, ACM, pp. 1–12.

Demers, A., Petersen, K., Spreitzer, M., Ferry, D., Theimer, M. & Welch, B. (1994). The bayou architecture: Support for data sharing among mobile users, *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, IEEE, pp. 2–7.

Dippe, M. & Swensen, J. (1984). An adaptive subdivision algorithm and parallel architecture for realistic image synthesis, *ACM SIGGRAPH Computer Graphics*, Vol. 18, ACM, pp. 149–158.

Dong, Z., Kautz, J., Theobalt, C. & Seidel, H.-P. (2007). Interactive global illumination using implicit visibility, *Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on*, IEEE, pp. 77–86.

Drago, F., Myszkowski, K., Annen, T. & Chiba, N. (2003). Adaptive logarithmic mapping for displaying high contrast scenes, *Computer Graphics Forum*, Vol. 22, Wiley Online Library, pp. 419–426.

Durand, F. & Dorsey, J. (2002). Fast bilateral filtering for the display of high-dynamic-range images, *ACM Transactions on Graphics (TOG)*, Vol. 21, ACM, pp. 257–266.

Dustdar, S., Guo, Y., Satzger, B. & Truong, H.-L. (2011). Principles of elastic processes, *IEEE Internet Computing* **15**(5): 66–71.

Dutre, P., Bala, K., Bekaert, P. & Shirley, P. (2003). *Advanced global illumination*, Vol. 11, AK Peters.

Faure, H. (1982). Discrépance de suites associées à un système de numération (en dimension s), *Acta Arithmetica* **41**(4): 337–351.

Fidge, C. J. (1988). Timestamps in message-passing systems that preserve the partial ordering, *Proceedings of the 11th Australian Computer Science Conference*, Vol. 10, pp. 56–66.

Freeman, E., Hupfer, S. & Arnold, K. (1999). *JavaSpaces principles, patterns, and practice*, Addison-Wesley Professional.

Fuchs, H., Kedem, Z. M. & Naylor, B. F. (1980). On visible surface generation by a priori tree structures, *ACM Siggraph Computer Graphics*, Vol. 14, ACM, pp. 124–133.

Fujimoto, A., Tanaka, T. & Iwata, K. (1986). Arts: Accelerated ray-tracing system, *Computer Graphics and Applications, IEEE* **6**(4): 16–26.

Gaudet, S., Hobson, R., Chilka, P. & Calvert, T. (1988). Multiprocessor experiments for high-speed ray tracing, *ACM Transactions on Graphics (TOG)* **7**(3): 151–179.

Glassner, A. (1988). Space subdivision for fast ray tracing, *Computer Graphics and Applications, IEEE* **4**(4): 15–22.

Gonzalez-Morcillo, C., Weiss, G., Vallejo, D., Jimenez-Linares, L. & Castro-Schez, J. J. (2010). A multiagent architecture for 3d rendering optimization, *Applied Artificial Intelligence* **24**(4): 313–349.

Gooding, S. L., Arns, L., Smith, P. & Tillotson, J. (2006). Implementation of a distributed rendering environment for the teragrid, *Challenges of Large Applications in Distributed Environments, 2006 IEEE*, IEEE, pp. 13–21.

Goral, C. M., Torrance, K. E., Greenberg, D. P. & Battaile, B. (1984). Modeling the interaction of light between diffuse surfaces, *ACM SIGGRAPH Computer Graphics*, Vol. 18, ACM, pp. 213–222.

Gouraud, H. (1971). Continuous shading of curved surfaces, *Computers, IEEE Transactions on* **100**(6): 623–629.

Green, S. A. & Paddon, D. J. (1990). A highly flexible multiprocessor solution for ray tracing, *The visual computer* **6**(2): 62–73.

Greenberg, D. P., Torrance, K. E., Shirley, P., Arvo, J., Lafortune, E., Ferwerda, J. A., Walter, B., Trumbore, B., Pattanaik, S. & Foo, S.-C. (1997). A framework for realistic image synthesis, *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., pp. 477–494.

Greger, G., Shirley, P., Hubbard, P. M. & Greenberg, D. P. (1998). The irradiance volume, *Computer Graphics and Applications, IEEE* **18**(2): 32–43.

Hachisuka, T., Ogaki, S. & Jensen, H. W. (2008). Progressive photon mapping, *ACM Transactions on Graphics (TOG)*, Vol. 27, ACM, p. 130.

Haeberli, P. & Akeley, K. (1990). The accumulation buffer: hardware support for high-quality rendering, *ACM SIGGRAPH Computer Graphics* **24**(4): 309–318.

Halton, J. H. (1964). Algorithm 247: Radical-inverse quasi-random point sequence, *Communications of the ACM* **7**(12): 701–702.

Hammersley, J. M. (1960). Monte carlo methods for solving multivariable problems, *Annals of the New York Academy of Sciences* **86**(3): 844–874.

Happa, J., Bashford-Rogers, T., Wilkie, A., Artusi, A., Debattista, K. & Chalmers, A. (2012). Cultural heritage predictive rendering, *Computer Graphics Forum* .

Happa, J., Mudge, M., Debattista, K., Artusi, A., Gonçalves, A. & Chalmers, A. (2010). Illuminating the past: state of the art, *Virtual reality* **14**(3): 155–182.

Heckbert, P. (1992). Discontinuity meshing for radiosity, *Third Eurographics Workshop on Rendering*, pp. 203–226.

Heckbert, P. S. (1990). Adaptive radiosity textures for bidirectional ray tracing, *ACM SIGGRAPH Computer Graphics*, Vol. 24, ACM, pp. 145–154.

Heirich, A. & Arvo, J. (1998). A competitive analysis of load balancing strategies for parallel ray tracing, *The Journal of Supercomputing* **12**(1-2): 57–68.

Immel, D. S., Cohen, M. F. & Greenberg, D. P. (1986). A radiosity method for non-diffuse environments, *ACM SIGGRAPH Computer Graphics*, Vol. 20, ACM, pp. 133–142.

Iwasaki, K., Dobashi, Y., Yoshimoto, F. & Nishita, T. (2007). Precomputed radiance transfer for dynamic scenes taking into account light interreflection, *Proceedings of the 18th Eurographics conference on Rendering Techniques*, Eurographics Association, pp. 35–44.

Jacobson, V., Frederick, R., Casner, S. & Schulzrinne, H. (2003). Rtp: A transport protocol for real-time applications.

Jarosz, W. (2008). *Efficient Monte Carlo Methods for Light Transport in Scattering Media*, PhD thesis, UC San Diego.

Jelasity, M. & van Steen, M. (2002). Large-scale newscast computing on the internet.

Jensen, H. W. (2001). *Realistic image synthesis using photon mapping*, AK Peters, Ltd.

Kajiya, J. T. (1986). The rendering equation, *ACM Siggraph Computer Graphics*, Vol. 20, ACM, pp. 143–150.

Kaplanyan, A. & Dachsbacher, C. (2010). Cascaded light propagation volumes for real-time indirect illumination, *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, ACM, pp. 99–107.

Kay, T. L. & Kajiya, J. T. (1986). Ray tracing complex scenes, *ACM Siggraph Computer Graphics*, Vol. 20, ACM, pp. 269–278.

Keates, M. J. & Hubbold, R. J. (1995). Interactive ray tracing on a virtual shared-memory parallel computer, *Computer Graphics Forum*, Vol. 14, Wiley Online Library, pp. 189–202.

Keller, A. (1997). Instant radiosity, *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., pp. 49–56.

Kermack, W. O. & McKendrick, A. G. (1932). Contributions to the mathematical theory of epidemics. ii. the problem of endemicity, *Proceedings of the Royal society of London. Series A* **138**(834): 55–83.

Knecht, M. (2007). State of the art report on ambient occlusion, *Vienna Institute of Technology, Technical Report* .

Kneebone, R. (2003). Simulation in surgical training: educational issues and practical implications, *Medical education* **37**(3): 267–277.

Koholka, R., Mayer, H. & Goller, A. (1999). Mpi-parallelized radiance on sgi cow and smp, *Parallel computation*, Springer, pp. 549–558.

Kollig, T. & Keller, A. (2006). Illumination in the presence of weak singularities, *Monte Carlo and Quasi-Monte Carlo Methods 2004*, Springer, pp. 245–257.

Kopf, J., Cohen, M. F., Lischinski, D. & Uyttendaele, M. (2007). Joint bilateral upsampling, *ACM Transactions on Graphics (TOG)*, Vol. 26, ACM, p. 96.

Krivanek, J. & Gautron, P. (2009). Practical global illumination with irradiance caching, *Synthesis lectures on computer graphics and animation* **4**(1): 1–148.

Krzyzanowski, P. (n.d.). Lectures on distributed systems: Clock synchronization, *Lecture notes, Rutgers University Computer Science* **416**.

Labidi, M., Tang, B., Fedak, G., Khemakem, M. & Jemni, M. (2012). Scheduling data on data-driven master/worker platform, *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2012 13th International Conference on*, IEEE, pp. 593–598.

Lafortune, E. P., Foo, S.-C., Torrance, K. E. & Greenberg, D. P. (1997). Non-linear approximation of reflectance functions, *Proceedings of the 24th*

annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., pp. 117–126.

Lafortune, E. P. & Willems, Y. D. (1993). Bi-directional path tracing, *Proceedings of CompuGraphics*, Vol. 93, pp. 145–153.

Lamport, L. (1974). A new solution of dijkstra's concurrent programming problem, *Communications of the ACM* **17**(8): 453–455.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system, *Communications of the ACM* **21**(7): 558–565.

Langer, M. S. & Bülthoff, H. H. (1999). Depth discrimination from shading under diffuse lighting., *Perception* **29**(6): 649–660.

Larson, G. W., Shakespeare, R., Ehrlich, C., Mardaljevic, J., Phillips, E. & Apian-Bennewitz, P. (1998). *Rendering with radiance: the art and science of lighting visualization*, Morgan Kaufmann San Francisco, CA.

László, S.-K. (1999). Monte-carlo methods in global illumination, *Institute of Compute Graphics, Technische Universität Wien* .

Lehmann, M. A. (2014). Lzf compression library (liblzf).
URL: *http://www.goof.com/pcg/marc/liblzf.html*

Lewis, R. R. (1994). Making shaders more physically plausible, *Computer Graphics Forum*, Vol. 13, Wiley Online Library, pp. 109–120.

Li, K. (1988). Ivy: A shared virtual memory system for parallel computing., *ICPP (2)*, pp. 94–101.

Li, K. & Hudak, P. (1989). Memory coherence in shared virtual memory systems, *ACM Transactions on Computer Systems (TOCS)* **7**(4): 321–359.

Lin, T. T. & Slater, M. (1991). Stochastic ray tracing using simd processor arrays, *The Visual Computer* **7**(4): 187–199.

Luft, T., Colditz, C. & Deussen, O. (2006). Image enhancement by unsharp masking the depth buffer, *ACM Transactions on Graphics* **25**: 1206–1213.

MacDonald, J. D. & Booth, K. S. (1990). Heuristics for ray tracing using space subdivision, *The Visual Computer* **6**(3): 153–166.

Mantiuk, R., Daly, S. J., Myszkowski, K. & Seidel, H.-P. (2005). Predicting visible differences in high dynamic range images: model and its calibration, *Electronic Imaging 2005*, International Society for Optics and Photonics, pp. 204–214.

Manzano, M., Hernández, J. A., Uruenña, M. & Calle, E. (2012). An empirical study of cloud gaming, *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, IEEE Press, p. 17.

Mara, M., Luebke, D. & McGuire, M. (2013). Toward practical real-time photon mapping: efficient gpu density estimation, *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, pp. 71–78.

Marks, S., Windsor, J. & Wünsche, B. (2007). Evaluation of game engines for simulated surgical training, *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, ACM, pp. 273–280.

Matusik, W. (2003). *A data-driven reflectance model*, PhD thesis, Massachusetts Institute of Technology.

McGuire, M. & Luebke, D. (2009). Hardware-accelerated global illumination by image space photon mapping, *Proceedings of the 2009 ACM SIGGRAPH/EuroGraphics conference on High Performance Graphics*, ACM, New York, NY, USA.
URL: *http://graphics.cs.williams.edu/papers/PhotonHPG09/*

Mell, P. & Grance, T. (2011). The nist definition of cloud computing.

Metropolis, N. & Ulam, S. (1949). The monte carlo method, *Journal of the American statistical association* **44**(247): 335–341.

Mitchell, J., McTaggart, G. & Green, C. (2006). Shading in valve's source engine, *ACM SIGGRAPH 2006 Courses*, ACM, pp. 129–142.

Mittring, M. (2007). Finding next gen: Cryengine 2, *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, ACM, New York, NY, USA, pp. 97–121.
URL: *http://doi.acm.org/10.1145/1281500.1281671*

Mosberger, D. (1993). Memory consistency models, *ACM SIGOPS Operating Systems Review* **27**(1): 18–26.

Muuss, M. J. (1995). To wards real-time ray-tracing of combinatorial solid geometric models, *Proceedings of BRL-CAD symposium*, Citeseer.

Myszkowski, K., Tawara, T., Akamine, H. & Seidel, H.-P. (2001). Perception-guided global illumination solution for animation rendering, *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, pp. 221–230.

Nemoto, K. & Omachi, T. (1986). An adaptive subdivision by sliding boundary surfaces, *Graphics Interface*, pp. 43–48.

Nicodemus, F. E. (1965). Directional reflectance and emissivity of an opaque surface, *Applied Optics* **4**(7): 767–773.

Niederreiter, H. (1988). Low-discrepancy and low-dispersion sequences, *Journal of number theory* **30**(1): 51–70.

Nishita, T. & Nakamae, E. (1985). Continuous tone representation of three-dimensional objects taking account of shadows and interreflection, *ACM SIGGRAPH Computer Graphics*, Vol. 19, ACM, pp. 23–30.

Nygren, N., Denzinger, J., Stephenson, B. & Aycock, J. (2011). User-preference-based automated level generation for platform games, *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, IEEE, pp. 55–62.

OnLive (2014).
URL: *http://www.onlive.com*

Pajak, D., Herzog, R., Eisemann, E., Myszkowski, K. & Seidel, H.-P. (2011). Scalable remote rendering with depth and motion-flow augmented streaming, *Computer Graphics Forum*, Vol. 30, Wiley Online Library, pp. 415–424.

Pál, L., Oláh-Gál, R. & Makó, Z. (2009). Shepard interpolation with stationary points, *Acta Univ. Sapientiae* **1**(1): 5–13.

Parker, S., Martin, W., Sloan, P.-P. J., Shirley, P., Smits, B. & Hansen, C. (1999). Interactive ray tracing, *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, I3D '99, ACM, New York, NY, USA, pp. 119–126.
URL: *http://doi.acm.org/10.1145/300523.300537*

Parker, S., Shirley, P., Livnat, Y., Hansen, C. & Sloan, P.-P. (1998). Interactive ray tracing for isosurface rendering, *Proceedings of the conference on Visualization'98*, IEEE Computer Society Press, pp. 233–238.

Patoli, M., Gkion, M., Al-Barakati, A., Zhang, W., Newbury, P. & White, M. (2009a). An open source grid based render farm for blender 3d, *Power Systems Conference and Exposition, 2009. PSCE'09. IEEE/PES*, IEEE, pp. 1–6.

Patoli, Z., Gkion, M., Al-Barakati, A., Zhang, W., Newbury, P. & White, M. (2009b). How to build an open source render farm based on desktop grid computing, *Wireless Networks, Information Processing and Systems*, Springer, pp. 268–278.

Pharr, M. & Humphreys, G. (2010). *Physically based rendering: From theory to implementation*, Morgan Kaufmann.

Phong, B. T. (1975). Illumination for computer generated pictures, *Communications of the ACM* **18**(6): 311–317.

Pitot, P. (1993). The voxar project (parallel ray-tracing), *Computer Graphics and Applications, IEEE* **13**(1): 27–33.

Pittel, B. (1987). On spreading a rumor, *SIAM Journal on Applied Mathematics* **47**(1): 213–223.

PlayStation Now (2014).
URL: *https://www.playstation.com/en-us/explore/psnow*

Plunkett, D. J. & Bailey, M. J. (1985). The vectorization of a ray-tracing algorithm for improved execution speed, *Computer Graphics and Applications, IEEE* **5**(8): 52–60.

Priol, T. & Bouatouch, K. (1989). Static load balancing for a parallel ray tracing on a mimd hypercube, *The Visual Computer* **5**(1-2): 109–119.

Ramos, J. M., González-Morcillo, C., Fernández, D. V. & López-López, L. (2009). Yafrid-ng: A peer to peer architecture for physically based rendering, *CEIG 09-Congreso Espanol de Informatica Grafica*, The Eurographics Association, pp. 227–230.

Rangel-Kuoppa, R., Avilés-Cruz, C. & Mould, D. (2003). Distributed 3d rendering system in a multi-agent platform, *Computer Science, 2003. ENC 2003. Proceedings of the Fourth Mexican International Conference on*, IEEE, pp. 168–175.

Reinhard, E., Chalmers, A. & Jansen, F. W. (1998). Overview of parallel photorealistic graphics, *Proceedings of Eurographics*, Vol. 98.

Reisman, A., Gotsman, C. & Schuster, A. (2000). Interactive-rate animation generation by parallel progressive ray-tracing on distributed-memory machines, *Journal of Parallel and Distributed Computing* **60**(9): 1074–1102.

renderRocket (2014).
URL: *http://www.renderrocket.com/features/*

Ritschel, T., Grosch, T., Kim, M. H., Seidel, H.-P., Dachsbacher, C. & Kautz, J. (2008). Imperfect shadow maps for efficient computation of indirect illumination, *ACM Transactions on Graphics (TOG)*, Vol. 27, ACM, p. 129.

Ritschel, T., Grosch, T. & Seidel, H.-P. (2009). Approximating dynamic global illumination in image space, *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, pp. 75–82.

Robertson, D., Campbell, K., Lau, S. & Ligocki, T. (1999). Parallelization of radiance for real time interactive lighting visualization walkthroughs, *Supercomputing, ACM/IEEE 1999 Conference*, IEEE, pp. 61–61.

Saito, T. & Takahashi, T. (1990). Comprehensible rendering of 3-d shapes, *ACM SIGGRAPH Computer Graphics*, Vol. 24, ACM, pp. 197–206.

Salmon, J. & Goldsmith, J. (1989). A hypercube ray-tracer, *Proceedings of the third conference on Hypercube concurrent computers and applications-Volume 2*, ACM, pp. 1194–1206.

Scherson, I. D. & Caspary, E. (1988). Multiprocessing for ray tracing: A hierarchical self-balancing approach, *The Visual Computer* **4**(4): 188–196.

Schlick, C. (1994). An inexpensive brdf model for physically-based rendering, *Computer graphics forum*, Vol. 13, Wiley Online Library, pp. 233–246.

Schlick, C. (1995). Quantization techniques for visualization of high dynamic range pictures, *Photorealistic Rendering Techniques*, Springer, pp. 7–20.

Shepard, D. (1968). A two-dimensional interpolation function for irregularly-spaced data, *Proceedings of the 1968 23rd ACM national conference*, ACM, pp. 517–524.

Shirley, P., Ashikhmin, M. & Marschner, S. (2009). *Fundamentals of computer graphics*, CRC Press.

Sillion, F. & Puech, C. (1989). A general two-pass method integrating specular and diffuse reflection, *ACM SIGGRAPH Computer Graphics*, Vol. 23, ACM, pp. 335–344.

Sillion, F. X., Puech, C. *et al.* (1994). *Radiosity and global illumination*, Vol. 11, Springer.

Sloan, P.-P., Govindaraju, N. K., Nowrouzezahrai, D. & Snyder, J. (2007). Image-based proxy accumulation for real-time soft global illumination, *Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on*, IEEE, pp. 97–105.

Sloan, P.-P., Kautz, J. & Snyder, J. (2002). Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments, *ACM Transactions on Graphics (TOG)*, Vol. 21, ACM, pp. 527–536.

Smith, T. & Guild, J. (1931). The cie colorimetric standards and their use, *Transactions of the Optical Society* **33**(3): 73.

Sobol, I. M. (1967). On the distribution of points in a cube and the approximate evaluation of integrals, *USSR Computational Mathematics and Mathematical Physics* **7**(4): 86–112.

Sobol, I. M. (1994). *A primer for the Monte Carlo method*, CRC press.

Spina, S., Debattista, K., Bugeja, K. & Chalmers, A. (2012). Fast scalable k-nn computation for very large point clouds, *Theory and Practice of Computer Graphics*, The Eurographics Association, pp. 85–92.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications, *ACM SIGCOMM Computer Communication Review*, Vol. 31, ACM, pp. 149–160.

Sundstedt, V., Chalmers, A. & Martinez, P. (2004). High fidelity reconstruction of the ancient egyptian temple of kalabsha, *Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, ACM, pp. 107–113.

Tabellion, E. & Lamorlette, A. (2004). An approximate global illumination system for computer generated films, *ACM Transactions on Graphics (TOG)*, Vol. 23, ACM, pp. 469–476.

Torrance, K. E. & Sparrow, E. M. (1967). Theory for off-specular reflection from roughened surfaces, *JOSA* **57**(9): 1105–1112.

Van der Corput, J. (1936). *Verteilungsfunktionen: Mitteilg 5*, NV Noord-Hollandsche Uitgevers Maatschappij.

Veach, E. & Guibas, L. J. (1997). Metropolis light transport, *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., pp. 65–76.

Voulgaris, S., Gavidia, D. & Van Steen, M. (2005). Cyclon: Inexpensive membership management for unstructured p2p overlays, *Journal of Network and Systems Management* **13**(2): 197–217.

Wald, I., Benthin, C. & Slusallek, P. (2002). A simple and practical method for interactive ray tracing of dynamic scenes, *Submitted for publication, also available as a technical report at http://graphics. cs. uni-sb. de/Publications* .

Wald, I., Kollig, T., Benthin, C., Keller, A. & Slusallek, P. (2002). Interactive global illumination using fast ray tracing, *Proceedings of the 13th Eurographics workshop on Rendering*, Eurographics Association, pp. 15–24.

Wald, I., Mark, W. R., Günther, J., Boulos, S., Ize, T., Hunt, W., Parker, S. G. & Shirley, P. (2009). State of the art in ray tracing animated scenes, *Computer Graphics Forum*, Vol. 28, Wiley Online Library, pp. 1691–1722.

Wald, I., Slusallek, P. & Benthin, C. (2001). *Interactive distributed ray tracing of highly complex models*, Springer.

Wald, I., Slusallek, P., Benthin, C. & Wagner, M. (2001). Interactive rendering with coherent ray tracing, *Computer graphics forum*, Vol. 20, Wiley Online Library, pp. 153–165.

Walker, D. W. & Dongarra, J. J. (1996). Mpi: a standard message passing interface, *Supercomputer* **12**: 56–68.

Wallace, J. R., Cohen, M. F. & Greenberg, D. P. (1987). A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods, *SIGGRAPH Comput. Graph.* **21**(4): 311–320.

Walter, B. (2005). Notes on the ward brdf, *Proceedings of Technical Report PCG-05–06, Cornell Program of Computer Graphics* .

Wang, R., Wang, R., Zhou, K., Pan, M. & Bao, H. (2009). An efficient gpu-based approach for interactive global illumination, *ACM Transactions on Graphics (TOG)* **28**(3): 91.

Ward, G. J. (1992). Measuring and modeling anisotropic reflection, *ACM SIGGRAPH Computer Graphics* **26**(2): 265–272.

Ward, G. J. (1994). The radiance lighting simulation and rendering system, *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM, pp. 459–472.

Ward, G. J., Rubinstein, F. M. & Clear, R. D. (1988). A ray tracing solution for diffuse interreflection, *ACM SIGGRAPH Computer Graphics* **22**(4): 85–92.

Whitted, T. (1980). An improved illumination model for shaded display, *Communications* .

Williams, L. (1978). Casting curved shadows on curved surfaces, *ACM Siggraph Computer Graphics*, Vol. 12, ACM, pp. 270–274.

Woodwark, J. (1984). A multiprocessor architecture for viewing solid models, *Displays* **5**(2): 97–103.

Yan, W., Culp, C. & Graf, R. (2011). Integrating bim and gaming for real-time interactive architectural visualization, *Automation in Construction* **20**(4): 446–458.

Yannakakis, G. N. & Togelius, J. (2011). Experience-driven procedural content generation, *Affective Computing, IEEE Transactions on* **2**(3): 147–161.

Yu, I., Cox, A., Kim, M. H., Ritschel, T., Grosch, T., Dachsbacher, C. & Kautz, J. (2009). Perceptual influence of approximate visibility in indirect illumination, *ACM Transactions on Applied Perception (TAP)* **6**(4): 24.

Zhao, B. Y., Kubiatowicz, J., Joseph, A. D. *et al.* (2001). Tapestry: An infrastructure for fault-tolerant wide-area location and routing.